Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Adaptar Abordagem KAOS para Especificar Linhas de Produtos de Software

Por
Dagmar do Rosário Mendes Cristóvão Baptista

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção de grau de Mestre em 21 de Abril de 2009

Orientador
Prof. Doutor João Baptista da Silva Araújo Júnior

Co-Orientadora
Prof. Dra. Carla Silva

Lisboa
2009

Agradecimentos

Começo por agradecer às minhas duas princesas Andreia e Ariana pela paciência, compreensão, amor, carinho e dedicação que tiveram ao longo desta fase. Quero agradecer também aos meus pais João Baptista e Marcelina Baptista, aos meus irmãos Juvenal, Franklin, Alvim e Nadjedine, e ao resto da família que mesmo estando longe apoiaram-me e incentivaram-me para a realização desta dissertação.

Agradeço também aos meus professores e orientadores desta dissertação Dr. João Baptista da Silva Araújo Júnior e Dr.ª Carla Taciana Lima Lourenço Silva Schuenemann, por me terem aceite e orientado ao longo da realização desta dissertação. Quero agradecer à minha colega Sandra pelo apoio e incentivo durante a dissertação. Expresso também o meu agradecimento a todos os professores do Departamento de Informática da Faculdade de Ciências e de Tecnologia da Universidade Nova de Lisboa, que sem eles nada disto seria possível.

Quero agradecer a todos os meus colegas e amigos de infância, entre eles, Luizito, Gika, Bambi, Vado, Tuca, Neno e outros, e também aos FCKs Alex, Furia, Flip, Cen, Ademar, Fxt, Pinha, Sansão, André Gil, Ivo, Magneticfield, Miguel, Oz, Bobcat, Kremlin, Dudu, Marco e Cátia.

Por último quero agradecer a todos que por algum motivo estiveram contra a realização desta dissertação.

Obrigado a todos.

Resumo

A investigação da Engenharia de Requisitos (ER) para as Linhas de Produtos de Software (LPSs) tem explorado as maneiras onde se pode definir apropriadamente os artefactos base, capaz de servir como base para a derivação rentável de produtos para utilizadores individuais. Uma LPS é um conjunto ou um grupo de produtos que têm características em comum, mas também varia em determinadas características. A modelação de *feature* é uma técnica chave para capturar pontos comuns e variáveis em sistema de famílias e linhas de produtos. Portanto, devido à complexidade do desenvolvimento da LPS, a ER é muito importante para gerir esta complexidade.

A Engenharia de Requisitos Orientada a Objectivos (EROO) tem sido usada para desenvolver sistemas complexos e algumas abordagens têm sido desenvolvidas, tais como KAOS. Um objectivo *(goal)* é um propósito que um sistema é suposto alcançar.

Entretanto, o uso do KAOS para descrever uma LPS ainda não foi explorado suficientemente. Mas o modelo de objectivos fornece propriedades para a identificação da variabilidade na fase inicial dos requisitos.

Deste modo, o objectivo desta dissertação é adaptar a abordagem KAOS para modelar as LPSs, de modo a fornecer uma abordagem orientada a objectivos expressiva para o desenvolvimento de uma LPS.

Abstract

Research on requirements engineering for software product lines (SPLs) has been exploring ways by which one can define core assets capable of serving as the basis for cost-effective derivation of products for individual users. A product line (PL) is a set or group of products that have features in common, but also vary in certain features. Feature modeling is a key technique for capturing commonalities and variabilities in system families and product lines. Therefore, due to the complexity of product line development, requirements engineering is much important to manage this complexity.

Goal-oriented requirements engineering has been used to develop complex systems and some approaches have been developed such as KAOS. A goal is a purpose that a system is supposed to achieve.

However, the use of KAOS to describe PL has not been explored sufficiently yet. But goal models provide features to identify variability at the early requirements phase.

Therefore, the objective of this dissertation is to tailor KAOS approach to model product lines in order to provide a more expressive goal-oriented approach for product line development.

Acrónimos

Lista de acrónimos usados nesta dissertação.

Acrónimos	Significado
ER	Engenharia de Requisitos
EROO	Engenharia de Requisitos Orientada a Objectivos
LPS	Linha de Produtos de Software
ELPS	Engenharia de Linha de Produtos de Software
DFD	Diagrama de Fluxos Diagramas
DER	Diagrama de Entidades e Relações
KAOS	Keep All Objectives Satisfied
GRL	Goal-Oriented Requirements Language
NFR	Non-Functional Requirements
NFRF	Non-Functional Requirements Framework
UML	Unifield Modeling Language
UMLT	Unifield Modeling Language Transformation
FODA	Feature-Oriented Domanin Analysis
FORM	Feature-Oriented Reuse Method
MATA	Modeling Aspects using a Transformation Approach
SVS	Serviço de Vigilância de Saúde

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

1 Introdução

1.1 Engenharia de Requisitos

A Engenharia de Requisitos (ER) cobre todas as actividades envolvidas na descoberta, documentação e manutenção de um conjunto de requisitos para um determinado sistema [1]. O uso do termo 'engenharia' implica que técnicas sistemáticas e repetitivas podem ser usadas para assegurar que os requisitos de um sistema são completos, consistentes, relevantes, etc. Os requisitos são definidos durante a fase inicial do desenvolvimento de um sistema como uma especificação que deve ser implementada. Eles descrevem como o sistema deve comportar-se, além de propriedades e atributos que o sistema deve apresentar. Os requisitos podem ser também uma restrição no processo de desenvolvimento do sistema [2].

Os requisitos reflectem as necessidades dos diferentes *stakeholders* de um sistema a desenvolver. Estes *stakeholders* são pessoas que serão afectadas pelo sistema e que têm influência directa ou indirectamente nos requisitos do sistema [2].

A engenharia de requisitos inclui um conjunto de actividades, tais como [1, 2]:

- Identificação de requisitos – Nesta actividade é efectuada o descobrimento dos requisitos de um futuro sistema. O objectivo é encontrar, sobre o domínio da aplicação, quais os serviços que o sistema deve proporcionar, o desempenho solicitado do sistema, as restrições, entre outros. Estes requisitos são capturados através de excelentes fontes de informação, tais como entrevistas, questionários, pesquisas e modelos de processos, documentação de sistemas existentes na organização, entre outros;

- Análise de requisitos - A análise de requisitos é necessária para resolver os problemas da Identificação e alcançar um entendimento nas mudanças para os requisitos, uma vez que estes requisitos são elicitados de diferentes *stakeholders*;

- Documentação de requisitos – Nesta actividade são usados diferentes diagramas para descrever os requisitos em vários níveis de detalhe, tais como: Diagramas de Fluxos de Dados (DFD) para modelar dados que são processados por um sistema, em termos de entrada e saída; Diagramas de Entidades e Relações (DER) para relacionar as entidades do sistema; Diagrama de Casos de Uso para modelar os principais serviços de um sistema; entre outros;

- Validação de requisitos – Este processo consiste em demonstrar que os requisitos definem o sistema que o cliente quer. Ele tem como objectivo verificar o conjunto de requisitos definidos e descobrir os possíveis problemas destes requisitos. O processo envolve *stakeholders* do sistema e os engenheiros de requisitos. Estas validações são realizadas através de algumas técnicas, tais como, revisões de requisitos, prototipagem, geração de casos de teste, análise automatizada de consistência, entre outras;

- Gestão de requisitos – Permite gerir as mudanças nos requisitos durante o processo de engenharia de requisitos, e o processo de desenvolvimento de sistemas. Esta gestão assegura que a qualidade dos requisitos seja mantida.

Esta tese centra-se nas fases de Identificação e análise de requisitos. Os requisitos em ER podem ser categorizados em dois tipos principais: os requisitos funcionais e os requisitos não-funcionais [2].

Os requisitos funcionais descrevem os serviços que o sistema deve oferecer, como o sistema deve reagir a certas entradas, e como o sistema deve comportar-se em determinadas situações [1, 2].

Os requisitos não-funcionais referem-se aos atributos de qualidade de um sistema, isto é, as restrições de como os requisitos funcionais são implementados num sistema, tais como segurança, disponibilidade, tempo de resposta, fiabilidade, normas, desempenho, entre outras. Estes requisitos são propriedades ou qualidades que um sistema possui. Frequentemente, por haver conflitos entre os vários requisitos de qualidade existentes em um sistema, eles não podem ser satisfeitos na sua totalidade, levando a optar pelos mais importantes para o sistema a desenvolver [1, 2].

Quando se inicia o desenvolvimento de uma aplicação, é possível que os requisitos iniciais expressos pelos *stakeholders* sejam ambíguos, incompletos e inconsistentes. Deste modo, é

frequente recorrer às actividades do ER, para a produção de um documento de requisitos adequado, que posteriormente resulte num software de qualidade.

Existem várias abordagens de documentação de requisitos, tais como as orientadas a: objectivos (*goal-oriented*) [3-5]; pontos de vista (*viewpoint-oriented*) [6]; e casos de uso (*use case*) [7, 8]. Neste documento o maior foco estará centrado na engenharia de requisitos orientada a objectivos (*Goal – Oriented Requirements Engineering*).

Uma vez que o trabalho aborda as linhas de produtos de software, na secção seguinte é feita uma breve introdução às linhas de produtos.

1.2 Linhas de Produtos de software

Uma Linha de Produtos de Software (LPS) é um conjunto ou grupo de produtos que têm uma maioria de propriedades comuns e apenas variando em certas propriedades específicas. Desenvolver um grupo de produtos que têm uma maioria de propriedades em comum, contribui muito para reutilização em todas as fases do desenvolvimento de um sistema [9-11].

Uma LPS tem duas actividades importantes: desenvolvimento dos artefactos base (*core assets*) e o desenvolvimento do produto. Na actividade de desenvolvimento dos artefactos base, onde também é conhecida como engenharia de domínio, requisitos da família de produtos e potenciais membros desta família são identificados, analisados e uma *framework* de domínio reutilizável é desenvolvida. Esta *framework* contém cinco sub-processos importantes: a gestão do produto; a engenharia de requisito do domínio, o desenho do domínio; realização do domínio; e a verificação dos componentes reutilizáveis. No desenvolvimento do produto, também designado como engenharia de aplicação, a *framework* de domínio é instanciada para desenvolver aplicações individuais [9-11].

Devido à complexidade e à natureza extensiva do desenvolvimento das linhas de produto, o ER é muito importante para a sua prática. Aplicando as diversas técnicas da engenharia de requisitos no desenvolvimento das linhas de produto de software, facilita o seu desenvolvimento por parte dos diferentes *stakeholders* e permite a construção de linhas de produtos robustas [9, 10].

Neste documento é também evidenciado o modelo de *feature* para apoiar o desenvolvimento de uma LPS, uma vez que fornece propriedades que permite tratar das características comuns e variáveis existentes em uma linha de produtos.

1.3 Motivação

Quando utilizamos a engenharia de requisitos orientada a objectivos, onde um objectivo (*goal*) é aquilo que um sistema deve alcançar, é necessária a participação e compromisso dos *stakeholders* para ajudar na obtenção e gestão dos objectivos de um sistema em desenvolvimento. Estes objectivos são descrições de como o sistema deve comportar, ou uma propriedade de sistema [4, 12, 13].

Actualmente, o desenvolvimento de linhas de produtos de software tem se tornado mais comum. As linhas de produtos tiram partido da estratégia de reutilização, possuindo diversos benefícios, entre eles a diminuição do tempo de construção do produto para o mercado, o aumento da agilidade do mercado, aumento da satisfação do cliente, habilidade para efectuar produção de larga escala e mais eficiência na utilização dos recursos humanos [9, 10]. Entretanto, em geral nas abordagens existentes, a especificação dos requisitos do domínio e a justificação dos modelos de *features* carecem de uma abordagem de requisitos mais efectiva que ajude na descoberta das *features* da LPS.

As abordagens orientadas a objectivos, como por exemplo o KAOS [3, 5, 14], *framework* i* [15], GRL [16, 17], V-graph [18], entre outras, não se encontram ainda muito bem exploradas para linhas de produtos de software. Neste documento daremos mais ênfase a abordagem KAOS, com objectivo de adaptá-la para a especificação e integração do desenvolvimento de linhas de produtos de software.

1.4 Objectivos e Contribuições Esperadas

O objectivo deste trabalho é adaptar a abordagem KAOS para modelar linhas de produtos de software, de modo a fornecer uma abordagem de requisitos orientada a objectivos expressiva para o desenvolvimento de linhas de produto.

A abordagem KAOS não tem sido explorada suficientemente para as linhas de produtos de software. O modelo de objectivos tem a potencialidade de identificar a variabilidade nas fases iniciais dos requisitos que pode ser combinado com modelo de *feature*, bastante utilizado no desenvolvimento de LPS. Entretanto, este modelo pode-se tornar muito complexo em sistemas de larga escala e, em particular, em linhas de produto de software. Parte deste trabalho é endereçar este problema através de técnicas de modularização.

A contribuição deste trabalho é realçar os resultados produzidos por uma abordagem de requisitos orientada a objectivos, introduzindo um mecanismo mais eficiente da modularização. O alvo apontado é um requisito de sistema compreensível e mais envolvente.

1.5 Organização do documento

Além deste capítulo introdutório, este documento consiste em mais cinco capítulos:

Capítulo 2 – Engenharia de Requisitos Orientada a Objectivos

Neste capítulo são descritos os principais conceitos da engenharia de requisitos orientada a objectivos. Para além destes conceitos, são descritos também algumas abordagens orientadas a objectivos, tais como KAOS, GRL, i* e V-graph. Destas abordagens é focado com mais pormenor o modelo KAOS.

Capítulo 3 – Engenharia de Linha de Produtos de Software

Neste capítulo são relatadas as principais características desta abordagem e também o modelo de *features* que é uma técnica chave para capturar e controlar as *features* comuns e variáveis de uma linha de produtos.

Capítulo 4 – Trabalhos relacionados

Neste capítulo é descrito alguns trabalhos relacionados a abordagem em questão, bem como a sua contribuição de modo a dar uma visão mais ampla da respectiva área.

Capítulo 5 – Abordagem LPS-KAOS

Neste capítulo é descrito dois processos para a abordagem LPS-KAOS e aplicado um exemplo prático.

Capítulo 6 – Caso de estudo e comparação com outras abordagens

Neste capítulo é apresentado um caso de estudo para ilustrar a abordagem LPS-KAOS e a comparação com outras abordagens.

Capítulo 7 – Conclusão

Neste capítulo é apresentado a conclusão desta dissertação.

2 Engenharia de Requisitos Orientada a Objectivos

As mudanças organizacionais envolvem o desenvolvimento de sistemas computadorizados ou de reestruturação de processos de negócios, que são actividades intencionais conduzidas pelos objectivos das partes interessadas. A sua eficácia depende de se tomar boas decisões sobre quais objectivos devem ser levados a cabo, e na selecção de estratégias apropriadas para alcançar os objectivos desejados. Um objectivo (*Goal)* é um propósito que o sistema deve alcançar após a execução de uma acção [19]. Os *stakeholders*, durante o desenvolvimento de um sistema, desempenham um papel importante na área de engenharia de requisitos, visto que eles se encontram presentes durante o ciclo de vida do software [4, 12, 13].

A Engenharia de Requisitos está relacionada com a obtenção dos objectivos de alto nível para serem alcançados por sistemas previstos. Está também relacionada pelo refinamento destes objectivos e a sua utilização operacional em especificações de serviços e das restrições, e pela atribuição de responsabilidades para os requisitos resultantes aos agentes, tais como seres humanos, hardware e software [12].

É importante salientar que o conceito chave de objectivo encontra-se presente em toda e qualquer abordagem orientada a objectivos, de modo a haver uma melhor percepção dos aspectos existentes nesta abordagem, e uma melhor interacção entre os diferentes *stakeholders* na utilização da mesma.

Os objectivos são refinados em sub-objectivos, e decompostos em vários requisitos. Os requisitos relacionados com os objectivos podem ser funcionais e não-funcionais. Os funcionais reflectem as funcionalidades do sistema e os não-funcionais estão relacionados a um tipo de objectivos denominado por *softgoal,* que referem aos atributos de qualidade de um sistema [4, 12].

Existem várias abordagens orientadas a objectivos, tais como KAOS, i* *Framework*, NFR *Framework* [19], GRL, V-graph, entre outras, mas neste documento iremos dar ênfase ao método KAOS, que será o foco da dissertação.

2.1 KAOS

O KAOS [3, 5, 14] é uma abordagem da engenharia de requisitos orientada a objectivos (EROO) e, oferece uma *framework* para Identificação de requisitos e gestão de sistemas. KAOS significa "*Keep All Objectives Satisfied*", isto é, manter todos os objectivos satisfeitos [20]. Existem algumas ferramentas que servem de apoio a abordagem, tais como o *Objectiver* [21], que foi utilizado para ilustrar alguns exemplos neste documento.

A ideia chave por trás da abordagem KAOS é a construção de um modelo de requisitos, isto é, descrever o problema a ser resolvido e as restrições que devem ser cumpridas por qualquer solução fornecida [14].

Deste modo, o KAOS foi concebido para [14]:

- Ajustar as descrições de problemas, permitindo definir e manipular conceitos relevantes para a descrição do problema;
- Aperfeiçoar o processo de análise de problemas, fornecendo uma abordagem sistemática para descobrir e estruturar os requisitos;
- Clarificar as responsabilidades de todos os *stakeholders* envolvidos no projecto;
- Permitir que os engenheiros de desenvolvimento comuniquem-se facilmente e eficientemente sobre os requisitos;
- Evitar os requisitos irrelevantes e ambíguos;
- Escolher diferentes alternativas;
- Gerir conflitos.

2.1.1 Conceitos

Os objectivos são os conceitos mais importantes na abordagem KAOS e são propriedades desejadas do sistema que são expressas pelos *stakeholders* [14]. Eles podem ser formulados em diferentes níveis de abstracção [4, 14]. Os objectivos de um sistema podem ser recolhidos por meio de entrevistas aos utilizadores; por análise de sistemas existentes, interpretando os documentos técnicos disponíveis; entre outros [14].

Na abordagem KAOS, os objectivos podem ser decompostos através de ligações E (*AND*) e OU (*OR*), e estas denominam-se por refinamento. As ligações de refinamento *AND* relacionam um objectivo com um conjunto de sub-objectivos, isto significa que satisfazer todos os sub-objectivos no refinamento é a condição para satisfazer o objectivo. As ligações de refinamento *OR* relacionam um objectivo com um conjunto alternativo de refinamentos, isto significa que satisfazer um dos refinamentos é a condição para satisfazer o objectivo. A estrutura do refinamento *AND/OR* dos objectivos para um determinado sistema, pode ser representada através de um grafo acíclico dirigido [13].

Quando se fala de objectivos em KAOS, é necessário distingui-los de acordo com os diferentes tipos de categoria de requisitos, isto é, requisitos funcionais e os não-funcionais [14]. A Figura 2.1 ilustra um exemplo do refinamento de objectivos pertencentes a requisitos funcionais e a não funcionais.

Figura 2.1 - Exemplo da decomposição de objectivos funcionais e não funcionais [22].

Na abordagem KAOS, os objectivos são ilustrados em representações gráficas específicas como mostra a Figura 2.1. Assim no topo do grafo encontram-se os requisitos de negócio que

são representados como objectivos, e na base temos os requisitos de sistema. Os requisitos de negócio são expressos em termos do vocabulário dos *stakeholders* pertencentes ao desenvolvimento do sistema, ao passo que os requisitos de sistema para além de serem expressos como os de negócio, são também expressos em termos técnicos específicos que são introduzidos no modelo em locais onde são necessários [14].

Quando usamos a abordagem KAOS, podemos efectuar uma análise de cima para baixo (*Top - Down*) ou de baixo para cima (*Bottom - Up*). É aconselhável o uso de ambas análises ao mesmo tempo, pois permitirá aos analistas alcançarem um ponto onde alguns objectivos são modelados a partir dos objectivos mais detalhados para os gerais e outros a partir dos objectivos gerais para os mais detalhados [14].

2.1.2 Modelos do KAOS

O modelo de requisitos da abordagem KAOS é composto por quatro importantes sub-modelos [14]:

- Modelos de objectivos;
- Modelo de responsabilidades;
- Modelo de objectos;
- Modelo de operação.

Na realidade, além dos objectivos, as responsabilidades, os objectos e as operações de um sistema podem fazer parte do modelo KAOS em geral. Para se ter uma visão mais precisa dos conceitos e propriedades da abordagem KAOS, isto é, as potencialidades dos modelos propostos na abordagem, nada melhor do que um exemplo ilustrativo. Deste modo, considera-se o exemplo da "via verde" [23]:

"Pretende-se desenvolver um sistema que utiliza identificadores tipo "via verde" para aceder a parques de estacionamento e o abastecimento de combustível em estações de serviço. Os identificadores obtêm-se através de um simples processo de adesão, onde o cliente fornece os seus dados pessoais, do seu cartão de débito e ainda do veículo a registar. É necessário fazer a activação do identificador numa caixa de multibanco associando este ao cartão de débito.

Para aceder a um parque basta que o identificador seja colado no pára-brisas do veículo e aproximá-lo de uma das cancelas especiais, que se abrirá se ele for válido e acederá uma luz

verde. Para sair do parque o procedimento é semelhante. A quantia a pagar depende do tempo gasto no parque e é mostrada num visor (*display)* acoplado a uma cancela de saída. Ainda, qualquer reclamação relacionada ao sistema (e.g. erro na leitura do identificador seja na entrada ou na saída do veículo, ou erro no cálculo da quantia a ser debitada) pode ser feita a um funcionário do parque que deverá registá-la no sistema. O valor a pagar é enviado semanalmente pelo sistema ao banco onde o cliente tem a conta para os débitos. Se houver algum problema na transacção, o banco deve notificar o sistema e o departamento de contabilidade deverá enviar uma carta ao cliente.

Para abastecer basta também que o identificador colado no pára-brisas do veículo seja válido. Um veículo aderente, ao aproximar-se de uma das várias bombas de abastecimento automático, activa o sistema que acende uma luz verde. Para abastecer, o condutor deve inserir primeiro o código do cartão de débito associado ao identificador. O abastecimento termina quando o condutor coloca a mangueira de volta no suporte da bomba. O valor a pagar, que depende do tipo e quantidade de combustível seleccionado, é mostrado no visor da bomba. Uma vez terminada a operação o veículo pode seguir. O débito em conta é automático, portanto se a conta não tiver saldo suficiente o abastecimento não é autorizado."

Para este exemplo prático considerou-se os seguintes objectivos: efectuar adesão, activar identificador, entrada no parque, saída do parque e efectuar abastecimento na bomba de combustível. De modo a dar uma visão dos modelos em causa, apenas alguns deste goals irão ser utilizados neste capítulo, visto que os modelos completos serão apresentados no caso de estudo da dissertação. Foram também considerados os seguintes *softgoals*: disponibilidade, compatibilidade, segurança, tempo de resposta, correctude e o acesso múltiplo.

Para além dos objectivos e *softgoals*, o sistema possui alguns agentes, tais como, o cliente, o identificador, o sistema de controlo (parque e estacionamento) e o funcionário.

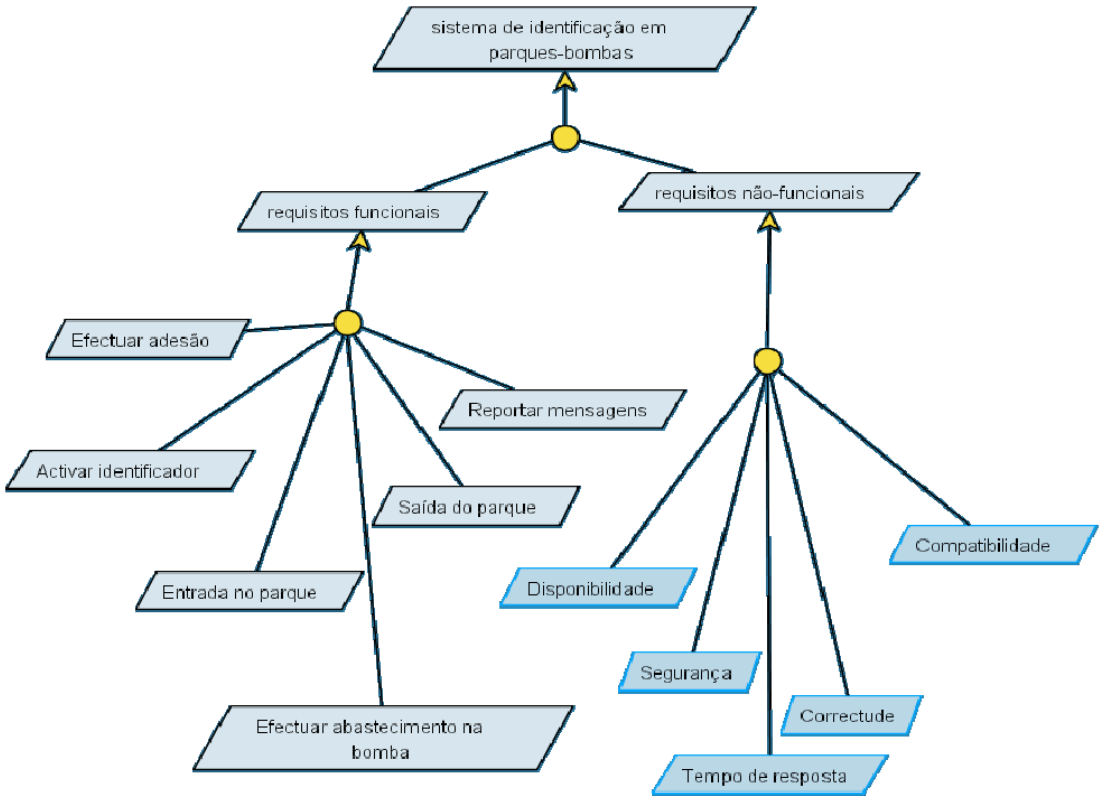
2.1.2.1 Modelo de objectivos

O modelo de objectivos [14] é composto por um conjunto de diagramas de objectivos inter-relacionados que foram reunidos para tentar resolver um determinado problema.

Nestes modelos existem paralelogramos e cada um deles representa um objectivo. Os círculos (em amarelo) nestes diagramas representam refinamentos que permitem decompor os objectivos de mais alto nível em uma lista de sub-objectivos que devem ser satisfeitos para alcançar o objectivo [14]. Na Figura 2.2 é ilustrado um modelo de objectivos, o qual é

composto por alguns objectivos, *softgoals* e refinamentos. Nesta figura o sistema encontra-se refinado em requisitos funcionais (objectivos) e não-funcionais (*softgoals*). As ligações que observamos entre os objectivos são de refinamento *AND*.

Figura 2.2 – Exemplo de um modelo de objectivos incompleto em que se pode encontrar objectivos, sub-objectivos e de refinamentos.



Um modelo de objectivos em KAOS é um diagrama dirigido (mais geral que uma árvore). Isto permite que um dado objectivo possa aparecer em diferentes diagramas para refinar diferentes objectivos de alto nível. Deste modo, é importante guardar no modelo todas as diferentes razões que justifiquem porquê um determinado objectivo é necessário e para que o modelo de objectivos seja completo é necessário satisfazer o seguinte critério de integridade [14]:

- Critério de integridade 1: Um modelo de objectivo é completo com respeito ao relacionamento de refinamento, se e só se, todo objectivo folha é uma expectativa, uma propriedade do domínio ou um requisito.

2.1.2.2 Modelo de responsabilidades

No modelo KAOS para além dos objectivos, é importante realçar também o conceito de agentes, que são humanos ou componentes automatizados que são responsáveis por alcançar

as expectativas e os requisitos. Um requisito é um objectivo posto sob a responsabilidade de um agente, e a ligação de refinamento para os requisitos é constituída por círculos vermelhos. Uma expectativa é um tipo de objectivo a ser alcançado por um agente que é parte do ambiente do sistema, que tem como ligação de refinamento círculos de cor rosa [14].

De acordo com [14], os requisitos são ilustrados nos diagramas por paralelogramos com bordas grossas. As expectativas são paralelogramos de cor amarela no seu interior e contém bordas grossas. Os agentes são representados por hexágono, como por exemplo o Cliente na Figura 2.3.

O modelo KAOS exige que os analistas associem cada requisito ou expectativa a um agente, o qual será responsável por alcançá-los [14]. Na Figura 2.3 é ilustrado um modelo de responsabilidade, que se encontra representado um agente (Cliente), que é responsável por quatro expectativas.

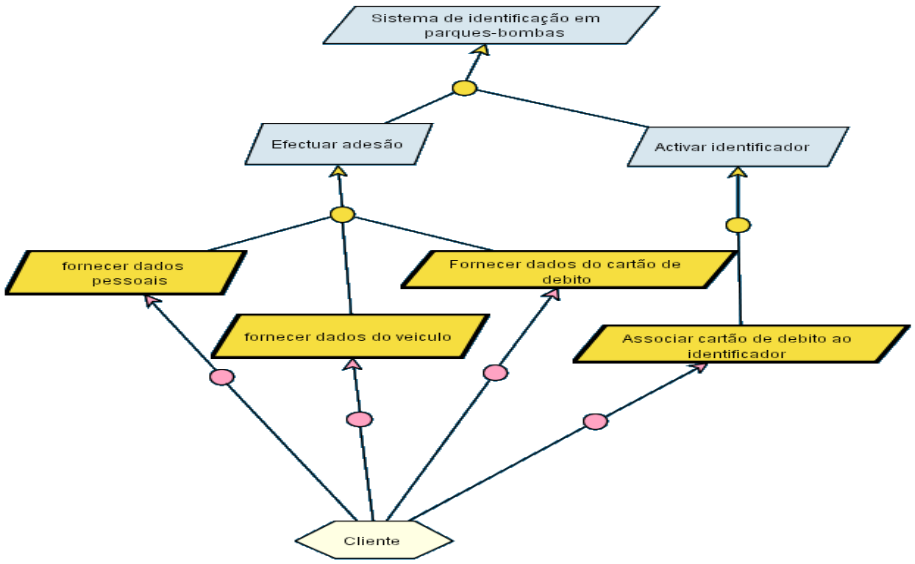


Figura 2.3 - Exemplo de um modelo de responsabilidade onde podemos destacar um agente cliente que é responsável por 4 expectativas.

Em muitos dos casos, vários agentes podem apontar a diferentes objectivos ao invés de apenas um. Esta diferença é feita entre estas duas possibilidades com o modelo KAOS, de modo que a atribuição seja usada quando um agente podem ser responsáveis por alguns requisitos ou algumas expectativas e a responsabilidade seja usada quando existe apenas um agente que é responsável por um requisito ou expectativa [14].

O modelo de responsabilidade [14] contém todos os diagramas de responsabilidade de um sistema. Estes diagramas descrevem para cada agente, os requisitos e expectativas dos quais

ele é responsável. Deste modo, para que um modelo seja completo com respeito o relacionamento de responsabilidade, é necessário cumprir o seguinte critério de integridade:

- Critério de integridade 2: Um modelo é dito completo com respeito a relação de responsabilidade, se e só se, todo o requisito e expectativa é posto sob a responsabilidade de um e só um agente.

Para construir estes diagramas, o analista examina os diferentes requisitos e expectativas no modelo de objectivos e associa um agente para cada um deles. Depois de todos os requisitos e expectativas se encontrarem associados a um agente, um diagrama é gerado para cada agente, listando os requisitos e expectativas associados a ele. Por definição o modelo de responsabilidade é derivado do modelo de objectivos [14].

2.1.2.3 Modelo de objectos

O modelo de objectos [14] é usado para definir e documentar os conceitos do domínio da aplicação que são relevantes no que diz respeito aos requisitos conhecidos e, para fornecer as restrições ao sistema que irá satisfazer os requisitos do sistema. Como parte deste modelo, encontramos objectos ligados ao domínio dos *stakeholders* e outros objectos introduzidos para expressar os requisitos ou restrições no sistema.

Os três tipos de conceitos que podem coexistir no modelo de objectos são [14]:

- As entidades representam objectos independentes e passivos. São independentes porque a sua descrição não precisa de referir a outros objectos do modelo. Podem ter atributos, cujos valores definem um conjunto de estados que a entidade poderá transaccionar. As entidades são passivas porque não podem executar operações.

- Os agentes representam objectos independentes e activos. Ser activo significa que os agentes executam operações. Usualmente, estas operações implicam a transição de estados das entidades.

- As associações permitem a ligação entre os objectos de um modelo. Existe uma associação entre dois ou mais objectos, se um objecto deve saber da existência ou executar um serviço de outro objecto.

A Figura 2.4 ilustra um modelo de objectos que descreve a relação dos objectos na entrada de um parque. A entidade máquina de entrada é relacionada com vários objectos, tais como: o

semáforo, a cancela, o sensor para detecção do identificador e o visor que permite mostrar informação.

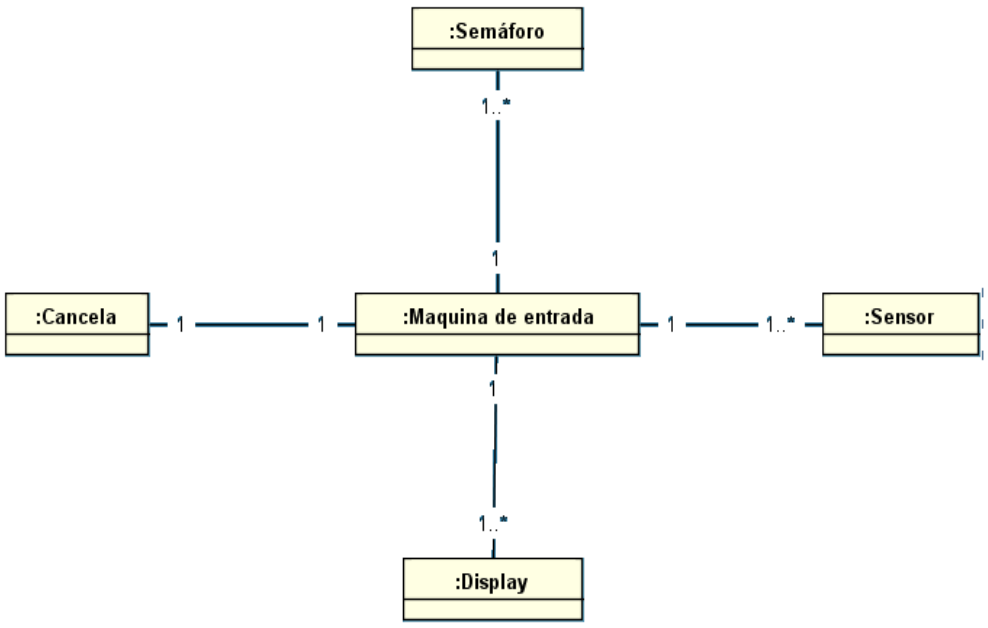


Figura 2.4 - Exemplo de um modelo de objecto.

O modelo de objectos no método KAOS é compatível com os diagramas de classes da Linguagem de Modelação Unificada (*Unifield Modeling Language* - UML) [7, 8], em que as entidades na abordagem KAOS correspondem às classes em UML e as associações correspondem às associações de multiplicidade *n* em UML. A herança também está disponível no modelo de objecto [14].

2.1.2.4 Modelo de operação

O modelo de operação [14] descreve todos os comportamentos que os agentes precisam de ter para cumprir os seus requisitos. Os comportamentos são expressos em termos de operações, que são executadas pelos agentes. As operações trabalham sobre as entidades (definidas no modelo de objecto) permitindo criar objectos, despoletar a transacção de estado de um objecto ou activar outras operações (por exemplo, envio de eventos).

Neste modelo, existem duas fontes de informação para identificar as operações [14]:

- Operações que podem ser expressas directamente por *stakeholders* durante as entrevistas;

- Operações que podem ser identificadas analisando todos os requisitos existentes. Elas ilustram como estes requisitos têm de ser realizados.

Um requisito do sistema pode ser operacionalizado por objecto (s), por comportamento (s) de agente (s) ou por uma combinação dos dois [14]:

- Os requisitos que descrevem propriedades estáticas do sistema são operacionalizados por objectos.

- Os requisitos que descrevem propriedades dinâmicas do sistema são operacionalizados por operações.

- Os requisitos que descrevem ambas as propriedades operacionalizam-se tanto com os objectos como com as operações.

Num modelo de operações, as operações são representadas na forma de uma elipse. Uma relação de operacionalização é representada por um círculo azul numa seta. Os objectos interessados são ligados às operações por meio de ligações de entrada e saída. Os eventos são representados por sinais de tráfego que são usados para as direcções indicadas. Eles podem ser externos ou produzidos por operações e também podem terminar ou começar operações [14]. Na Figura 2.5 é ilustrado um exemplo de um modelo de operação. No caso da entrada no parque de estacionamento após a verificação do identificador (válido), o sistema de controlo despoleta uma acção que visa a execução (*performance*) de uma operação para a abertura da cancela e acender a luz do semáforo, para a entrada do veículo que corresponde ao identificador lido.

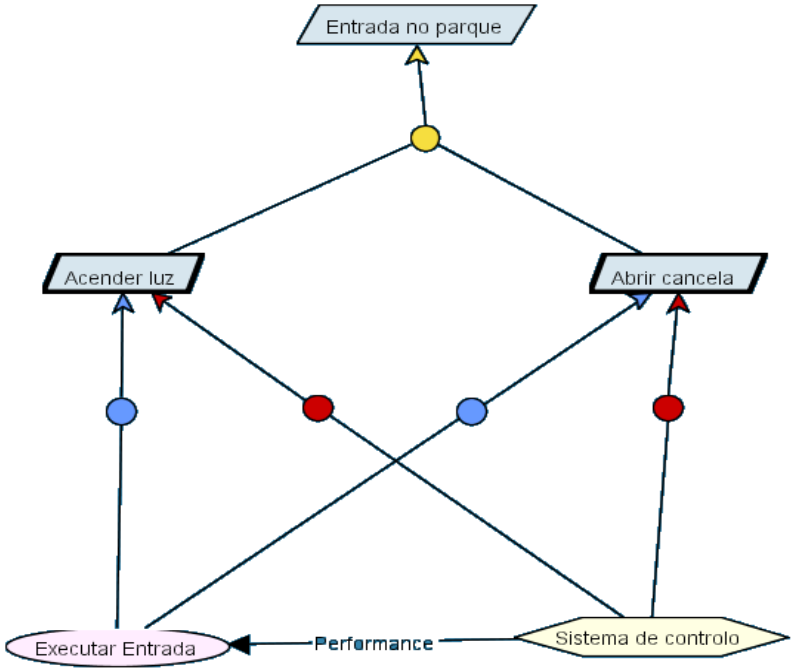


Figura 2.5 - Exemplo de um modelo de operação.

Quando falamos em operações, estamos a falar em processamento de dados e de um fluxo de controlo destes mesmos dados. A este processo que envolve a análise dos dados e de controlo, denomina-se de diagrama de processo. Este diagrama deve executar de acordo com os seguintes critérios de integridade [14]:

- Critério de integridade 3: Para ser completo, um diagrama de processo deve especificar: os agentes que executam as operações e os dados de entrada e saída para cada operação.

- Critério de integridade 4: Para ser completo, um diagrama de processo deve especificar quando é que as operações são executadas.

As operações da abordagem KAOS são usadas para operacionalizar os requisitos do sistema, e limitam o espaço das soluções que podem ser usados pelos engenheiros de desenvolvimentos para desenhar o sistema. Esta operacionalização dos requisitos permite uma ligação entre o espaço de descrição do problema (objectivos, requisitos e objectos de domínio) e o espaço de descrição da solução (operações que representam comportamento do agente, e objectos que interagem com estas operações). Um bom documento de requisitos descreve o problema tão completo quanto possível, onde a especificação de uma solução é limitada apenas para o que é realmente necessário. Assim, surge mais o critério de integridade para o modelo de operações [14]:

- Critério de integridade 5: Todas as operações devem ser justificadas pela existência de alguns requisitos, isto é, pelo uso das ligações de operacionalização.

Os relacionamentos nesta abordagem, isto é, refinamento, operacionalização, responsabilidade e outros, fornecem rastreabilidade entre todas as peças do modelo. Isto dá também flexibilidade ao analista para explorar o modelo (a partir das operações para os objectivos ou vice-versa) [14].

2.1.3 Obstáculos e conflitos

Os obstáculos são situações que violam um objectivo, um requisito ou uma expectativa. Neste caso, diz-se que o obstáculo "obstrui" o objectivo, requisito ou expectativa. O tratamento de obstáculos permite que os analistas identifiquem e localizem as circunstâncias excepcionais durante a engenharia de requisitos, a fim de produzir por exemplo, requisitos robustos ou até mesmo novos requisitos para evitar ou reduzir o impacto dos obstáculos, resultando num software de confiança. Os obstáculos impedem com que os objectivos de um sistema sejam alcançados. Lidar com obstáculos é muito importante para a construção de sistemas seguros e robustos [14].

A análise dos obstáculos na abordagem KAOS é uma actividade orientada a objectivos, que começa com a exploração do modelo de objectivo e com a negação de cada objectivo. Por exemplo um objectivo G é refinado em objectivos G1, …, Gn, em que G1, …, Gn todos juntos implicam em G, isto é, eles são uma condição suficiente para satisfazer G. Embora saibamos que se o objectivo G é violado, foi porque pelo menos um dos seus sub-objectivos foi violado [14].

Considerando o modelo KAOS como uma árvore no tratamento de um obstáculo, cada obstáculo folha é examinado com os peritos do domínio, para verificar se vale a pena considerá-lo na análise de obstáculo. Isto permite aos analistas eliminarem o espaço dos obstáculos e focar apenas nos obstáculos mais importantes existentes. Existem várias maneiras de identificar os obstáculos, começando por considerar falha de componentes ou o comportamento das pessoas por caminhos inesperados, com más intenções ou porque algumas pessoas têm capacidades limitadas. Uma vez identificadas as condições de ocorrência dos obstáculos, o analista pode solucionar o problema de diferentes maneiras [14]:

- Um primeiro caminho consiste em definir novos requisitos que previnam a ocorrência do obstáculo. Por exemplo, no requisito com obstáculo, adicionar refinamentos com novos requisitos de modo a que estes requisitos sejam a solução do obstáculo em causa.

- O segundo soluciona um obstáculo, restaurando o objectivo obstruído. Por exemplo, quando um computador é bloqueado, podemos reiniciá-lo.

- O terceiro consiste em minimizar os efeitos dos obstáculos. Por exemplo num local em que temos dois elevadores e dez andares, e o 1º elevador não passa do 5º andar devido a problemas. Podemos utilizar o 1º elevador até ao 5º andar. Deste modo, minimizamos os efeitos existentes de maneira que o 1º elevador não fique totalmente parado.

A Figura 2.6 ilustra a noção de obstáculo e a resolução do mesmo. No caso do objectivo "entrada no parque de estacionamento", quando o identificador é detectado pelo sensor da entrada do parque, o sistema reage automaticamente. Assim, o sistema necessita de validar o identificador e acender uma luz do semáforo para explicitar a validade do identificador, registar a hora de entrada para posteriormente determinar o tempo de estadia do veículo no parque e por último abrir a cancela. Por exemplo, se a cancela estiver avariada (obstáculo), propõe-se que o sistema resolva este obstáculo através da abertura manual da cancela por parte do funcionário do parque e assim permitir a entrada do veículo. Se o sensor estiver avariado, o sistema possibilita ao Cliente resolver o obstáculo com a retirada de um bilhete e para o caso das luzes do semáforo estiverem fundidas, o obstáculo é solucionado através de uma mensagem de voz na entrada do parque.

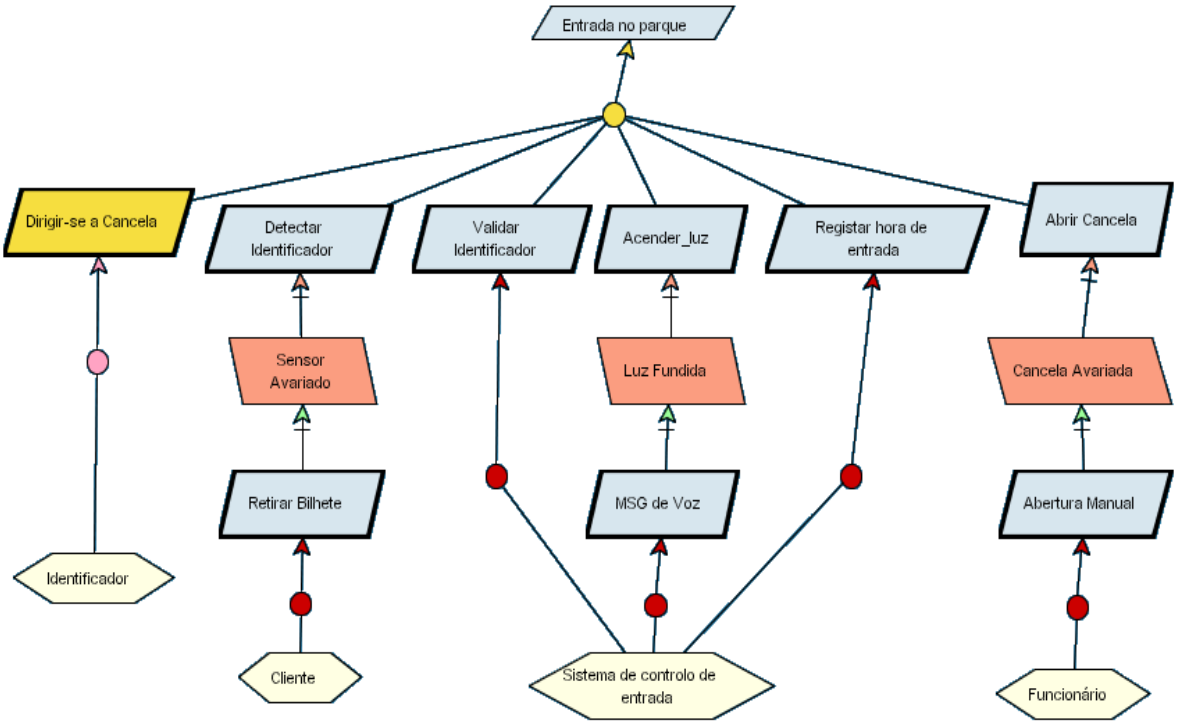


Figura 2.6 - Exemplo de identificação de obstáculo e sua possível resolução.

Quando se fala de conflitos, estamos a focar propriamente de problemas entre requisitos não-funcionais, isto é, *softgoals*. Para o caso do sistema em estudo, temos vários *softgoals,* tais como o tempo de resposta, segurança, disponibilidade, entre outros. Por exemplo, os *softgoals* segurança e tempo de resposta têm conflito, visto que elevados níveis de segurança diminui o tempo de resposta do sistema. Também podemos ter conflitos entre a correctude e tempo de resposta, visto que a verificação se os dados se encontram de forma correcta ou se são válidos, influência negativamente no tempo de resposta do sistema, e outros conflitos. Quando os conflitos e os obstáculos são detectados, existe uma negociação entre os diferentes *stakeholders* para a resolução dos mesmos, o que implica nas seguintes opções [14]:

- Selecção de alternativas.
- Reavaliação de prioridades.
- Revisão dos requisitos

É de relevante importância a detecção de conflitos o mais cedo possível no ciclo de vida do software de modo a não causar outros problemas, como por exemplo no sistema do parque de

estacionamento, um tempo de resposta muito alto pode levar a congestionamento nas entradas ou saídas do parque.

Para a nossa abordagem, vamos utilizar essencialmente o modelo de objectivos, onde vai ser demonstrado ser suficiente para especificar as partes comuns e variáveis de uma LPS.

Na secção seguinte serão descritas algumas abordagens orientadas a objectivos.

2.2 Outras Abordagens orientadas a objectivos

2.2.1 Framework i*

A abordagem i* [5, 24] é uma *framework* de modelação orientada a agentes e objectivos que pode ser usada para engenharia de requisitos, reestruturação de processos de negócio, análise de impacto organizacional e modelação de processos de software. Durante a fase inicial de requisitos, a *framework* i* é usada para modelar o ambiente do sistema. Isto facilita a análise do domínio, permitindo que o modelador represente esquematicamente os *stakeholders* do sistema, os seus objectivos e os seus relacionamentos. O analista pode, consequentemente, visualizar os processos actuais na organização e examinar o raciocínio por de trás destes processos.

O conceito central em i* é o de actor intencional. Os actores dentro de uma organização são vistos como tendo propriedades intencionais, tais como objectivos, crenças, capacidades e compromissos. Eles dependem de outros actores para alcançar os objectivos, realizar determinadas tarefas e disponibilizar recursos. Uma vez que eles dependem de outros actores, tornam-se vulneráveis se os outros actores dos quais dependem não corresponderem as suas expectativas [5, 15, 24].

Os actores podem ser agentes, posições ou papéis. Os agentes são actores em concreto, isto é, sistemas ou humanos com capacidades específicas. Um papel é a função que um actor desempenha. Uma posição é um conjunto de papéis reconhecidos socialmente e representados por um agente. Esta divisão é especialmente útil ao analisar o contexto social para um sistema de software. As dependências entre actores são identificadas como intencionais se aparecem como um resultado de agentes que levam a cabo os seus objectivos. Existem quatro tipos de dependências em i* [5, 15, 24]:

- Dependência de objectivo – ocorre quando um actor (*depender*) depende de outro actor (*dependee*) para alcançar um certo estado do mundo;

- Dependência de tarefa – ocorre quando um actor depende de outro para que este realize uma tarefa;

- Dependência de recurso – ocorre quando um actor depende de outro para que uma entidade (física ou computacional) seja disponibilizada;

- Dependência de *softgoal* – ocorre quando um actor depende de outro para que este desempenhe alguma tarefa para que um *softgoal* seja parcial ou totalmente satisfeito.

Existem dois tipos de modelos básicos na abordagem i* [5, 15, 24]:

- Modelo de Dependência Estratégica (*Strategic Dependency* - SD) – Descreve as relações de dependência entre actores.

- Modelos de Razão Estratégica (*Strategic Rationale* - SR) - Exprime como os actores atingem os seus objectivos.

2.2.2 NFR Framework

O NFR *framework* (*Non-Functional Requirements Framework* - NFRF) [19] foi concebido para representar e analisar os objectivos não-funcionais. O conceito mais importante para o NFRF é o *softgoal*, o qual representa um objectivo que não tem uma definição clara nem um critério preciso para determinar como é satisfeito. Os *softgoals* são usados para representar os requisitos não-funcionais e são interdependentes uns dos outros, sendo classificados em satisfeitos e não satisfeitos. Estes relacionamentos de interdependências são usados para ver como um *softgoal* é satisfeito*,* uma vez que alguns dos outros *softgoals* existentes podem ou não ser satisfeitos.

O NFRF consiste de cinco importantes componentes: *softgoal,* interdependências, procedimento de avaliação, métodos e correlações [19].

Quando os *softgoals* são refinados em *softgoals* descendentes são criados uma relação "é uma" (*IsA*). Os *softgoals* descendentes contribuem para que o seu *softgoal* "pai" seja afectado positivamente ou negativamente, isto é, no caso de uma contribuição positiva, a satisfação de uma descendência implica a satisfação do "pai", enquanto a contribuição negativa levam a rejeição do "pai" [19].

O procedimento de avaliação é aplicado ao grafo de interdependência de *softgoal* (*Softgoal interdependency Graph*) para determinar o grau a qual um *softgoal* inicial é satisfeito com um

dado conjunto de decisões. Os métodos fornecem um conjunto de procedimentos para refinar um *softgoal* ou uma interdependência em um ou mais *softgoals* descendentes. Os requisitos não-funcionais podem entrar em conflitos ou apoiarem-se entre si. Os relacionamentos entre estes requisitos são chamados de correlações, que são usados para examinar o impacto dos *softgoals* durante a sua análise [19].

Em NFRF, os *softgoals* identificados precisam de ser catalogados (para reutilização futura) e organizados em tipos (diferentes requisitos não-funcionais), e também precisam de ser organizados em hierarquia de relacionamentos *IsA* que refinam o *softgoal* inicial [19].

2.2.3 V- graph

O modelo V-graph [18, 25] consiste de um requisito funcional (representado por um objectivo), um requisito não-funcional (representado por um *softgoal*) e uma tarefa que contribui para a satisfação de ambos os requisitos. Este modelo é um grafo de três níveis, em forma da letra V. Os dois vértices do topo do grafo V representam requisitos funcionais e não-funcionais respectivamente em termos de modelos de objectivos. O vértice da base do grafo representa uma tarefa que contribui para a satisfação dos requisitos. O modelo permite uma descrição de nós intencionais (objectivos e *softgoals*) e de nós operacionais (tarefas).

Cada elemento do modelo é composto por um tipo e por um tópico. O tópico define a informação contextual de um elemento (por exemplo, dados e comunicação) e o tipo define um objectivo ou uma tarefa, isto é, descreve um requisito funcional ou não-funcional genérico, tais como desempenho, segurança e rastreabilidade. O tópico também permite delimitar o ponto de ligação entre os objectivos/tarefas e *softgoals*. Existem dois tipos de ligações que permitem o relacionamento entre os requisitos e as tarefas. Estas ligações podem ser de contribuição (*and, or, make, help, unknown, hurt, break*) ou de correlação (*help, unknown, hurt, break*). As ligações *make* e *help* indicam contribuição positiva; o *unknown* indica que há um relacionamento, mais é desconhecido se for positivo ou negativo; e o *hurt* e o *break* indicam contribuição negativa [18, 25].

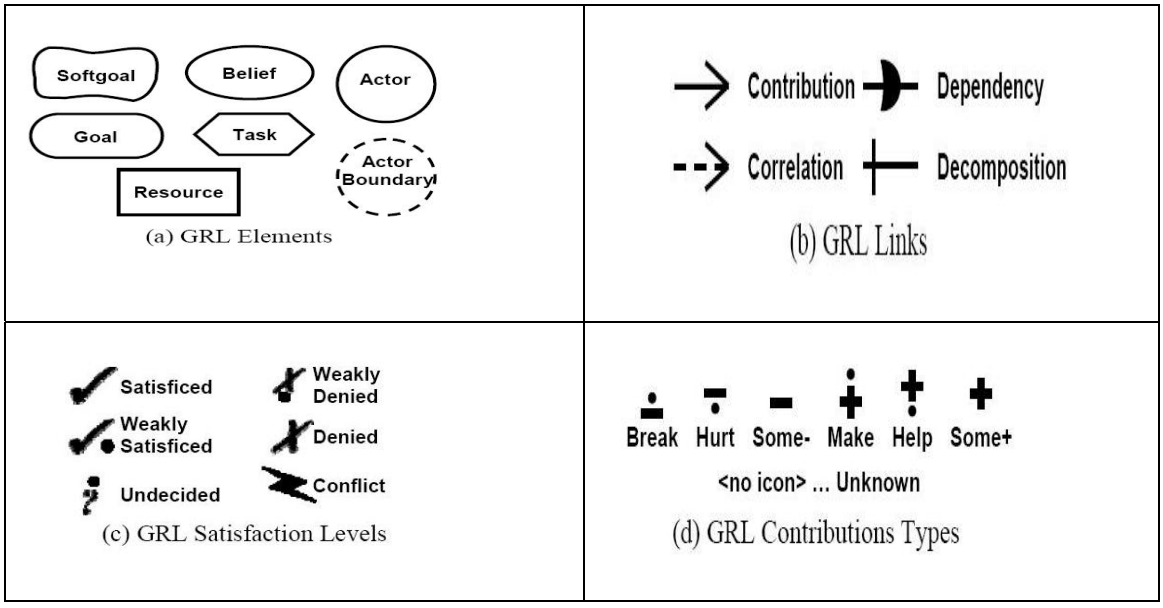
2.2.4 GRL

A Linguagem de Requisitos Orientada a Objectivo (*Goal - Oriented Requirement Language* - GRL) [16, 17] é uma notação de modelação visual que combina o NFR *framework* e a *framework* i* para apoiar o raciocínio dos modelos de objectivos.

A sintaxe do GRL é baseada na sintaxe do *framework* i*. Um grafo de objectivo do GRL é um grafo de refinamento E (*AND*) / OU (*OR*) de elementos intencionais (*softgoals*, objectivos, tarefas e recursos), onde opcionalmente reside na parte de dentro do limite de um actor. Um actor representa um *stakeholder* de um sistema. Este grafo mostra os objectivos de alto nível de negócio e os requisitos não-funcionais de interesse para um engenheiro do sistema, além das alternativas para alcançar estes elementos de alto nível. Um grafo de objectivos também permite documentar os raciocínios importantes para dos engenheiros de desenvolvimento. As tarefas representam soluções para os objectivos ou *softgoals*. De modo a serem alcançados ou completos, os *softgoals,* os objectivos e as tarefas podem requerer recursos para estarem disponíveis. Existem vários tipos de ligação entre os elementos de um grafo de objectivo, tais como, a decomposição, contribuição, correlação e a dependência [16, 17, 26].

Na Tabela 2.1 é apresentada a notação da abordagem GRL. Como se pode verificar, está notação difere do i* em alguns tipos, com por exemplo nos elementos (*Belief*) e nas ligações (*Correlation*).

Tabela 2.1 - Notação do GRL.



Do NFR *framework*, o GRL apropria-se da notação de um mecanismo de evolução que suporta o raciocínio sobre o grafo de objectivo. As decisões de um *stakeholder* são tipicamente documentadas no grafo de objectivo, pela tarefa de níveis de satisfação para alternativas. Basicamente, nesta definição inicial e as várias ligações com vários tipos de contribuições, os índices de satisfação são propagados para os objectivos de alto nível e para os requisitos não-funcionais dos *stakeholders* [16, 17, 26].

2.3 Resumo

Neste capítulo foi apresentado uma breve introdução para a engenharia de requisitos orientada a objectivos. Existem várias abordagens orientadas a objectivos, mas foram referidas apenas algumas abordagens, tais como KAOS, i*, NFRF, GRL e V-GRAPH. De entre estas abordagens referidas, o maior foco centrou-se na abordagem KAOS.

O objectivo é o conceito chave para toda e qualquer abordagem orientada a objectivo, isto permite uma melhor percepção dos aspectos existentes na abordagem, e uma melhor interacção entre os diferentes *stakeholders* do sistema a ser desenvolvido.

A abordagem KAOS é constituída por quatro modelos: o modelo de objectivos, o modelo de responsabilidade, o modelo de operação e o modelo de objectos. A ideia chave por trás da abordagem KAOS, é construir um modelo para os requisitos, isto é, para descrever o problema a ser resolvido, e as restrições que devem ser cumpridas por qualquer solução fornecida. A abordagem KAOS possui várias vantagens, tais como, permite a identificação dos requisitos na fase inicial do desenvolvimento de um sistema, permite que os objectivos de um sistema sejam refinados em requisitos e expectativas, permite a identificação da variabilidade na fase inicial da construção do sistema e permite também a identificação e resolução dos possíveis obstáculos e conflitos que possam surgir no sistema. Uma vez que a abordagem descreve a variabilidade na fase inicial da elaboração de um sistema, esta vai facilitar o desenvolvimento das LPSs, visto que esta variabilidade é um conceito muito importante nas linhas de produtos de software.

Uma vez que o objectivo do trabalho é a adaptação da abordagem KAOS para linha de produtos de software e neste capítulo foi a abordado os principais conceitos do modelo KAOS, é importante também focar as características da linha de produto de software. Deste modo, o próximo capítulo irá abordar os principais conceitos da engenharia de linhas de produtos de software.

3 Engenharia de Linhas de Produtos de Software

3.1 Descrição

Hoje em dia, é cada vez mais importante controlar produtos relacionados como membros de uma linha de produtos. O desenvolvimento rápido, a agilidade e a diferenciação são fundamentais para requerer um aumento de clientes [27].

Quando elaboramos a construção de um produto qualquer, pretendemos que o mesmo seja íntegro, robusto, de excelente desempenho, de baixo custo, e entre outros atributos de qualidade. É do nosso conhecimento que existem famílias de sistemas, em que os seus componentes podem ser agrupados consoante as semelhanças ou diferenças existentes em todos membros da família.

Uma família de sistemas é definida como um conjunto de programas que partilham funcionalidades comuns e preservam as funcionalidades específicas que variam de acordo com os membros da família [9, 10]. Deste modo, estas famílias de sistemas podem ser vistas como linhas de produtos de um determinado produto ou sistema.

De acordo com Clements e Northrop [9], uma Linha de Produto de Software (LPS) é um conjunto de sistemas intensivos que partilham e gerem um conjunto de propriedades comuns, satisfazendo um específico segmento de mercado necessário e que são desenvolvidos para um conjunto comum de artefactos base em um modo prescrito. Os artefactos base são aqueles artefactos reutilizáveis e recursos que formam a base para uma LPS. Entre os diversos artefactos base, a arquitectura do sistema é a chave na linha de produtos.

A Engenharia de Linhas de Produto de Software (ELPS) [10] contém métodos para o desenvolvimento de uma diversidade de produtos de software e sistemas intensivos de

software de baixo custo, em pouco tempo, e com alta qualidade. Segundo Pohl *et al*. [10], a ELPS é um paradigma para desenvolver aplicações de software (sistemas intensivos de software e produtos de software) usando plataformas e produção de larga escala.

Quando desenvolvemos uma linha de produtos através da ELPS, é necessária a criação de uma plataforma que colecciona todas as partes semelhantes. Esta plataforma (também denominada artefactos base) determina as especificações que distinguem não apenas as partes das linhas de produto, mas também como acomodar os clientes que desejam cada vez mais os produtos individualizados. No contexto de software uma plataforma é um conjunto de subsistemas de software e interfaces que formam uma estrutura comum, onde um conjunto de produtos derivados pode ser eficientemente desenvolvido e produzido [10].

Para facilitar a produção em larga escala, os artefactos usados nestes diferentes produtos têm de ser suficientemente adaptáveis para serem ajustados em diferentes sistemas produzidos na LPS, isto significa, que em todo o processo de desenvolvimento é necessário identificar e descrever onde os produtos na LPS diferem em termos das características que fornecem, os requisitos que cumprem, ou até em termos de arquitecturas subjacentes, etc. Deste modo, deve-se fornecer flexibilidade em todos estes artefactos para suportar a produção em larga escala, isto é, é possível definir os locais onde os produtos podem diferir. A esta flexibilidade que é uma pré-condição para a produção, é denominada de variabilidade no contexto de LPS, e é a base da produção em larga escala de produtos [10].

Uma vez que a LPS tira partido da reutilização estratégica de recursos, conseguem-se os seguintes benefícios [9]:

- Lucros de produtividade de larga escala;
- Diminuição do tempo de construção do produto para o mercado;
- Aumento da qualidade do produto;
- Diminuição dos riscos do produto;
- Aumento da agilidade do mercado;
- Aumento da satisfação do cliente;
- Mais eficiência na utilização de recursos humanos;

- Habilidade para efectuar produção de larga escala;

- Entre Outros.

3.2 Processos da ELPS

A ELPS encontra-se dividida em dois processos importantes para a construção de linhas de produtos [10]:

- Engenharia de domínio: Este processo é responsável pelo estabelecimento da plataforma reutilizável e permite definir os pontos comuns e variáveis de uma linha de produtos, ou seja, é o processo onde os pontos comuns e variáveis de uma LPS são definidas e realizadas. A plataforma consiste de todos os tipos de artefactos de software (requisitos, desenho, realização, testes, etc.) e a ligação entre estes artefactos facilita a reutilização sistemática e consistente.

- Engenharia de aplicação: Este processo é responsável pela produção das aplicações em uma linha de produtos, a partir da plataforma estabelecida na engenharia de domínio, ou seja, é o processo onde as aplicações da LPS são construídas através da reutilização dos artefactos de domínio, explorando as variabilidades de uma linha de produtos. Para além de explorar estas variabilidades, a engenharia de aplicação assegura a ligação correcta destas, de acordo com as aplicações necessárias.

Estes dois processos de desenvolvimento de uma linha de produtos também são conhecidos por duas actividades: o desenvolvimento de artefactos base (engenharia de domínio) e desenvolvimento do produto (engenharia de aplicação). Sendo estas actividades geridas pela actividade de gestão técnica e organizacional. Estas três actividades estão inter-relacionadas e são altamente interactivas [9]. A Figura 3.1 ilustra as três actividades essenciais no desenvolvimento de uma LPS.

Figura 3.1 - Actividades do processo da ELPS [9].

A vantagem para esta divisão dos processos da ELPS é a separação de dois conceitos para construir uma plataforma robusta e para construir aplicações específicas de utilizador em um curto espaço de tempo. Esta divisão também indica a separação em relação à variabilidade. A engenharia de domínio é responsável por assegurar que a variabilidade disponível seja apropriada para produzir as aplicações. Uma grande parte da engenharia de aplicação consiste na reutilização da plataforma definida na engenharia de domínio e na ligação da variabilidade como requerida pelas diferentes aplicações [10].

3.3 Framework da ELPS

De acordo Pohl *et al*. [10], esta *framework* é baseada na diferenciação entre os processos de engenharia a nível do domínio e a nível da aplicação proposta por Weiss e Lai [28]. A *framework* é ilustrada na Figura 3.2**.**

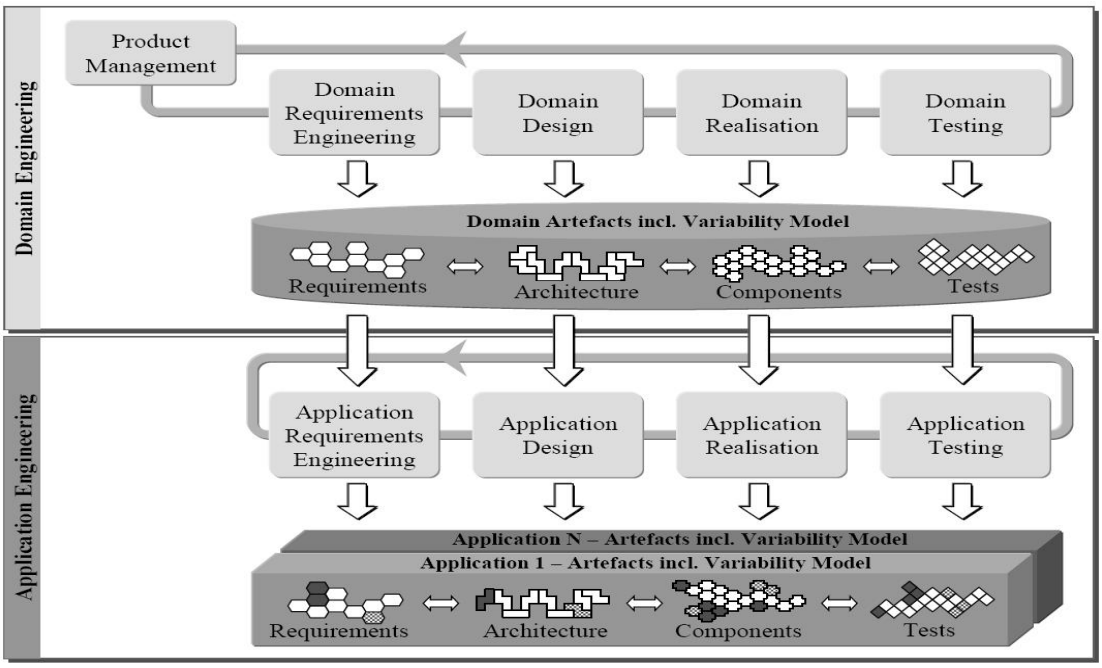


Figura 3.2 - Framework da engenharia de linha de produto de software [10].

Esta *framework* diferencia-se entre diferentes tipos de artefactos de desenvolvimento. Um artefacto de desenvolvimento é a saída (*output*) de um sub-processo da engenharia de domínio ou aplicação e estes incluem requisitos, arquitectura, componentes e testes. Estes artefactos podem ser os seguintes: Os artefactos do domínio que são artefactos de desenvolvimento reutilizável criados nos sub-processos de engenharia de domínio, e os artefactos da aplicação que são os artefactos de desenvolvimento de aplicações específicas da linha de produto [10].

O processo de engenharia de domínio é composto por 5 importantes sub-processos [10]:

- Gestão do produto – Este sub-processo permite lidar com os aspectos económicos de uma linha de produto de software e em particular com a estratégia de mercado. A principal função é a gestão da lista de documentos de um produto da companhia ou unidade de negócio. A ELPS permite definir o que está dentro da área da linha de produtos e o que está fora.

- Engenharia de requisito do domínio – Abrange todas as actividades para a descrição e documentação de requisitos comuns e variáveis de uma linha de produtos.

- Desenho do domínio – Este sub-processo contém todas as actividades para definir a arquitectura de referência de uma linha de produtos. Esta referência fornece uma arquitectura de alto nível semelhante para todas as aplicações da linha de produtos.

- Realização do domínio – Permite lidar com o desenho detalhado e a implementação de componentes reutilizável de software.

- Verificação ou teste do domínio – Esta fase é responsável pela validação e verificação dos componentes reutilizáveis. Este sub-processo testa os componentes contra as suas especificações, isto é, os requisitos, a arquitectura e os artefactos de desenho. Além disso, este sub-processo permite também desenvolver os artefactos de teste reutilizáveis para reduzir o esforço para verificação das aplicações.

Este processo de engenharia de domínio gera a plataforma incluindo os pontos comuns e as variedades da aplicação para servir de suporte a construção em larga escala [10].

De acordo com a Figura 3.2 que ilustra a *framework* da ELPS, o processo de engenharia de aplicação é composto pelos seguintes sub-processos [10]:

- Engenharia de requisitos da aplicação – Este sub-processo abrange todas as actividades para desenvolver a especificação dos requisitos de uma aplicação. A quantidade de artefactos reutilizáveis do domínio depende expressivamente dos requisitos da aplicação. Deste modo, o mais importante aqui é a detecção de variações entre os requisitos da aplicação e as capacidades disponíveis da plataforma.

- Desenho da aplicação – O desenho da aplicação ilustra as actividades para a produção da arquitectura da aplicação. Este sub-processo utiliza a arquitectura de referência para instanciar a arquitectura de uma aplicação. Permite também seleccionar e configurar as partes em causa da arquitectura de referência, e incorporar adaptações específicas da aplicação.

- Realização da aplicação – Neste sub-processo a aplicação considerada é criada. As principais funções deste sub-processo são a selecção e configuração dos componentes de software reutilizáveis, tal como a realização dos artefactos específicos da aplicação. Os artefactos reutilizáveis e os específicos da aplicação são reunidos para formar a aplicação.

- Verificação ou teste da aplicação – Este último processo inclui as actividades necessárias para validar e verificar uma aplicação contra as suas especificações.

Para além dos testes, outras técnicas que asseguram a qualidade do software são aplicadas na ELPS, principalmente inspecções rigorosas, análises críticas e um esforço conjunto de revisão com a finalidade de melhorar a qualidade do produto [10].

Para a elaboração deste trabalho, o maior foco irá centrar nos sub-processos de engenharia de requisitos do domínio e de engenharia de requisitos da aplicação, visto que abordam as fases iniciais do desenvolvimento de um produto.

3.4 Modelação de Variabilidade

Para modelar a variabilidade de uma linha de produtos, uma documentação adequada da variabilidade é necessária. A modelação da variabilidade é usada para gerir a complexidade que está envolvida nas actividades de desenvolvimento eficiente de uma diversidade de produtos. Estas actividades são as seguintes [29]:

- Os engenheiros de desenvolvimento identificam e descrevem a variabilidade das aplicações da linha de produtos em um modelo de *features* (propriedades) de forma que os clientes possam facilmente seleccionar as propriedades que desejam.

- Os engenheiros de desenvolvimento criam artefactos reutilizáveis de uma linha de produto de tal modo que estes artefactos são suficientemente adaptáveis para produzir eficientemente aplicações individuais a partir deles.

Na ELPS, a variabilidade é uma propriedade essencial dos artefactos do domínio. A variabilidade é introduzida durante a gestão do produto em sub-processos da engenharia de domínio, quando as características comuns e variáveis das aplicações da LPS são identificadas. Uma vez que os requisitos do domínio detalham as características definidas na gestão de produtos, a variabilidade é transferida para os requisitos do domínio. Similarmente isto é verdade para os sub-processos de desenho, de realização, e de teste [10].

Quando falamos de documentação da variabilidade, tem-se de responder as seguintes perguntas [10, 29]:

- O que é que varia? A resposta desta pergunta conduz para um item variável ou uma propriedade variável de um item, designado ponto de variação. Um ponto de variação é uma representação de um item variável dentro dos artefactos do domínio, enriquecido pela informação contextual;

- Como é que varia? Responder a esta questão identifica as instâncias possíveis de um item variável ou uma propriedade variável, denominadas variantes. Uma variante é uma representação de uma instância possível dentro do artefacto de domínio.

- Porquê é que varia? Esta pergunta conduz à razão por atrás de um ponto de variação ou variante.

- Para quem é o documento? Esta questão identifica o grupo alvo de um ponto de variação ou variante.

A variabilidade de uma LPS é modelada para permitir o desenvolvimento de aplicações personalizadas por reutilização predefinida, e por artefactos ajustáveis. Deste modo, a variabilidade de uma LPS permite distinguir as aplicações diferentes de uma linha de produto [10].

Existem várias abordagens para documentar a variabilidade de uma LPS, tais como a Análise de Domínio Orientada a *Feature* (*Feature-Oriented Domain Analysis* - FODA) [30], o Método de Reutilização Orientada a *Feature* (*Feature-Oriented Reuse Method* - FORM) [31], Modelo de *Features* (*Features Model*) [32], entre outras. Deste modo, para este trabalho será utilizado o modelo de *features*, visto ter sido bastante utilizado para a documentação de variabilidade da LPS, isto é, é uma abordagem chave para capturar variabilidades de um sistema.

3.5 Modelo de *Features*

Nesta secção é efectuada uma descrição do modelo de *features*.

3.5.1 Descrição

O modelo de *features* [30, 32, 33] foi introduzido como parte do método FODA e representa uma hierarquia de propriedades de conceitos do domínio. Uma *feature* é uma propriedade do sistema usada para distinguir os requisitos comuns dos requisitos variáveis de uma LPS.

De acordo Czarnecki et al [30, 33], o modelo de *features* é uma abordagem chave para capturar e controlar as *features* de um sistema. Nas fases iniciais do desenvolvimento de uma LPS, os modelos de *features* fornecem uma base para limitar as propriedades de uma família de sistemas, armazenando e acedendo informações, tais como: que *features* são importantes para entrar em um novo mercado ou permanecer em um mercado existente; que *features*

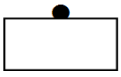
incorrem um risco tecnológico; qual o custo de desenvolvimento projectado para cada *feature*; e assim por diante. É importante realçar ainda que o modelo de *features* realiza um papel fundamental no desenvolvimento da arquitectura de uma família de sistemas, pois tem de realizar os pontos de variação especificados.

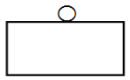
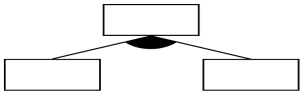
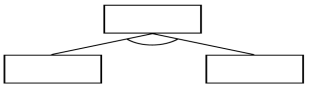
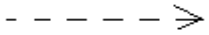
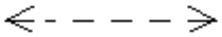
Na engenharia de domínio, o modelo de *features* pode conduzir à identificação e análise de requisitos. Por exemplo, saber quais *features* estão disponíveis na família de produtos pode ajudar aos clientes a decidir quais *features* o seu sistema deve apresentar. Além disso, saber quais *features* são fornecidas pela família de sistemas e quais *features* têm que ser adaptadas, ajuda a calcular melhor o tempo e o custo necessário para desenvolver o sistema [33].

Um modelo de *features* consiste de um ou mais diagramas de *features*. Um diagrama de *features* tem a forma de uma árvore, na qual a raiz da hierarquia é uma *feature* conceitual (por exemplo um sistema de software), que representa uma classe de soluções. Abaixo desta *feature* conceitual são estruturadas hierarquicamente *sub-features* que mostram propriedades refinadas do modelo [32, 33].

Um modelo de *features* representa uma vista abstracta sobre as propriedades de todas as ocorrências de um domínio. Segundo Czarnecki [33], toda a *feature* cobre um conjunto de requisitos. As *features* podem ser obrigatórias, opcionais, ou alternativas. As *features* obrigatórias fazem parte de todas as ocorrências do domínio, por definição. As opcionais podem não fazer parte do domínio e para serem controladas, as *features* alternativas relacionam entre duas ou mais *features* vizinhas na hierarquia. Nestas *features* alternativas, é necessário fazer a distinção de alternativas *OR* que permite mais do que uma *feature* e *XOR* que mostra a exclusão mútua [32, 33]. O modelo de *features* também introduz duas regras de decomposição: o "*requires*" que permite adicionar uma determinada *feature* a uma instância solicitada e o "*mutex-with*" permite fazer o inverso. A Tabela 3.1 ilustra as notações de um diagrama de *features.*

Tabela 3.1 - Notações de um diagrama de *features.*

Tipo de *feature*	**Notação**
Obrigatórias	

Opcionais	
Alternativa *OR*	
Alternativa *XOR*	
Requires	
Mutex-with	

Existem combinações entre as *features* obrigatórias e opcionais com as relações alternativas *OR* e *XOR*. Estas ligações podem levar a ambiguidades no diagrama de *features.* Para prevenir estas ambiguidades e permitir uma notação mais expressiva e poderosa para relações de *features* vizinhas, foram introduzidas as multiplicidades. Estas multiplicidades são similares às usadas em diagrama de classes em UML. Deste modo, diferentes multiplicidades são definidas [32, 33]:

- 0..1 – No máximo uma *feature* tem de ser escolhida do conjunto de *sub-features*.
- 1 – Exactamente uma *feature* tem de ser escolhida de um conjunto de *sub-features*.
- 0..* - Um número arbitrário de *feature* (ou nenhuma) tem de ser seleccionado do conjunto de *sub-features*.
- 1..* - Pelo menos uma *feature* tem de ser seleccionada do conjunto de *sub-features*.

Para além destas multiplicidades, existem outras cardinalidades para limitar as *features*. Uma multiplicidade X..Y, permite limitar um numero arbitrário de *feature* existente numa configuração possível. Por exemplo para X=1 e Y= 3 tem-se que existe pelo menos uma *feature* numa configuração e no máximo 3.

A Figura 3.3 ilustra um diagrama de *features* do exemplo da "Via Verde" usado neste documento para ilustrar as diversas abordagens apresentadas. Uma vez que o caso em estudo

foca um sistema que permite o estacionamento e abastecimento de veículos, a Figura 3.3 mostra o Serviço_Identificador como uma *feature* raiz. Esta *feature* é composta pelas *sub-features* obrigatórias (Identificador, Sensor e Débito) e opcionais (Estacionamento, Abastecimento e Portagem) e algumas destas *features* são descompostas em outras *features*. As obrigatórias aparecem em todas as configurações possíveis do sistema e as opcionais podem ou não aparecer.

No caso da *feature* Débito, esta é decomposta em duas *sub-features*: Débito_Quinzenal para o caso do Estacionamento e portagem e débito_Imediato para o caso do abastecimento, e encontram ligadas por "*requires*". Como se pode verificar, estas *sub-features* da *feature* Débito utilizam a relação *OR,* que permite a existência de apenas uma delas ou as duas em simultâneo, consoante a configuração das *features* opcionais (Estacionamento, Abastecimento e Portagem). Para o caso dos semáforos, havendo uma configuração possível apenas uma *feature* pode ser representada e não as duas em simultâneo, por isso utiliza-se a relação *XOR* de exclusão mútua. Mas isto não significa que o sistema só possa ter apenas estas *features,* outras podem ser adicionadas reutilizando as já existentes, por exemplo adicionar ao semáforo mais uma luz vermelha, apesar de estes sistemas utilizar apenas duas luzes.

Os modelos de *features* são usados para o desenvolvimento das LPSs e produtos de software específicos, ou seja, servem para definir produtos e configurações, para descrever as várias possibilidades de uma linha de produtos e para estabelecer novos produtos, além de adicionar novas propriedades para uma linha de produtos [32].

Este modelo de *features* foi elaborado na ferramenta *Captain feature* [34].

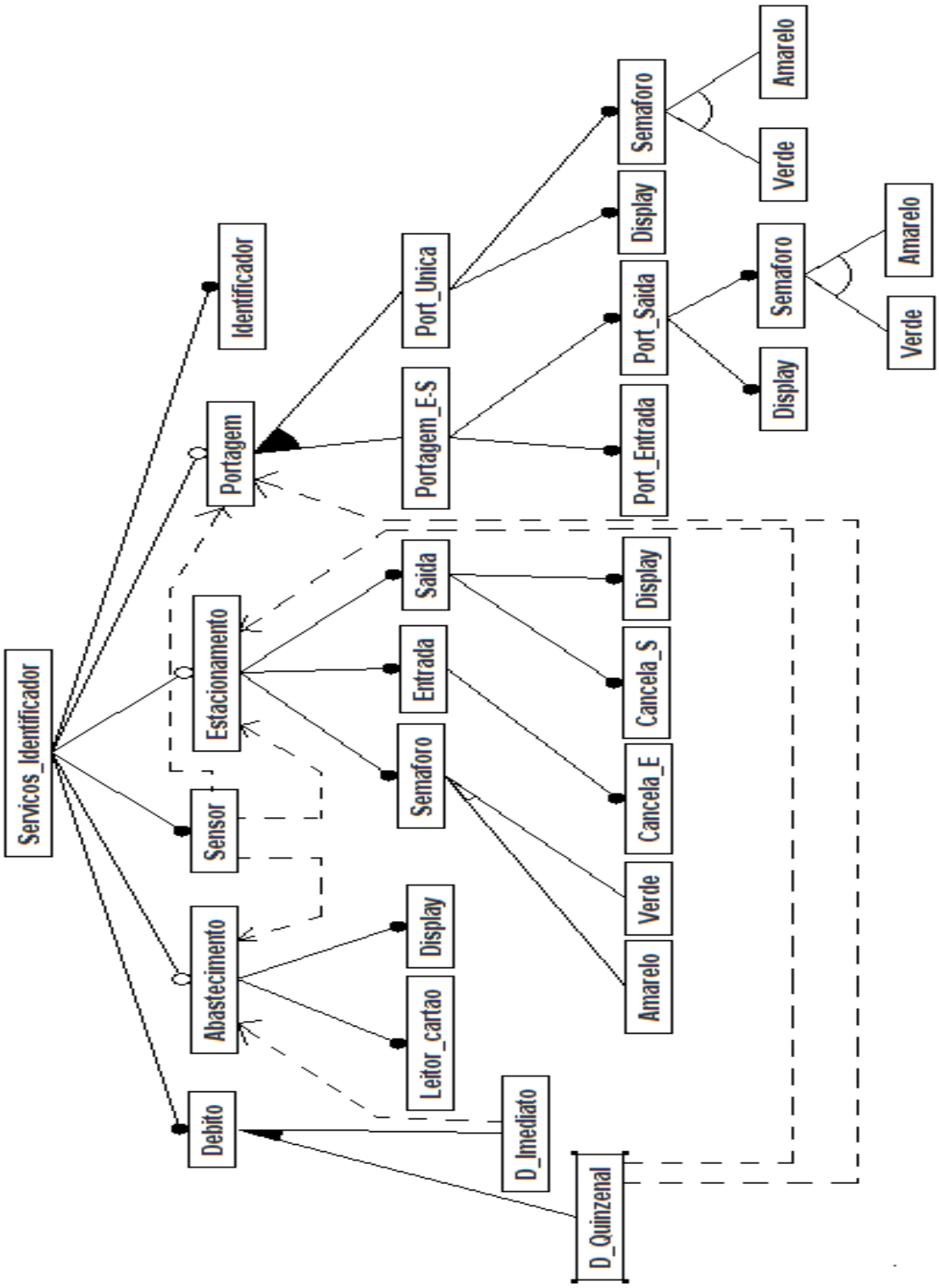


Figura 3.3 - Exemplo de um modelo de *features*.

3.6 Resumo

Neste capítulo foram discutidos os principais conceitos por de atrás da ELPS. O capítulo começou com uma breve explicação sobre as linhas de produtos, focando os conceitos de família de produtos e também alguns dos seus benefícios.

A ELPS é composta por três actividades essenciais: o desenvolvimento dos artefactos base, o desenvolvimento dos produtos e a actividade de gestão. As duas primeiras actividades são também designadas por dois processos: a engenharia do domínio e a engenharia da aplicação respectivamente. Cada uma das actividades realiza um papel importante na implementação de uma linha de produtos.

Estes processos são compostos por vários sub-processos. A engenharia do domínio é composta pela gestão do produto, pela engenharia de requisitos do domínio, pelo desenho do domínio, pela realização do domínio e pelo teste do domínio. A engenharia da aplicação é composta pela engenharia de requisitos da aplicação, pelo desenho da aplicação, pela realização da aplicação e pelo teste da aplicação.

Os artefactos base são o foco da LPS, e são também o bloco de construção para que novos produtos sejam manufacturados. O desenvolvimento de um produto específico é o principal propósito de uma linha de produtos. Finalmente as actividades de gestão são cruciais para adoptar uma abordagem da linha de produtos, porque todos recursos necessários são determinados nestas actividades.

Foi também apresentado o conceito de variabilidade, visto ser a propriedade central na ELPS.

Por último, neste capítulo, foi focado os conceitos mais importantes do modelo de *features* para documentar a variabilidade em uma LPS. Apesar de este modelo permitir uma boa documentação da variabilidade das LPSs, as várias abordagens que fazem o uso dos modelos de *features*, possuem algumas limitações, de entre elas, não identificam de forma sistemática as *features* de um sistema; não justificam nem documentam as escolhas das features para um produto individual; e não permitem a justificação sistemática das relações entre as *features*. Deste modo, procura-se solucionar estas limitações com a abordagem KAOS, visto que documentam a variabilidade de um sistema na fase inicial do desenvolvimento de um sistema.

No capítulo seguinte é apresentado alguns trabalhos relacionados com o tema desta dissertação de mestrado.

4 Trabalhos relacionados

As linhas de produtos de software e os modelos de *features* têm sido cada vez mais utilizados, e deste modo várias abordagens que englobam as LPSs e *features* têm sido propostas para resolver certos problemas existentes, integrando a abordagens de ER e em particular em EROO. Assim, este capítulo descreve algumas propostas que integram a ER e LPS.

4.1 Casos de uso e LPS

Uma abordagem para modelar casos de uso para linha de produtos de software é discutido em [35]. Segundo Gomaa [35], para especificar os requisitos funcionais das linhas de produto de software na abordagem de modelação de caso de uso, é necessário definir diferentes tipos de caso de uso: casos de uso central (*kernel use case*), que são necessários para todos os membros de uma linha de produto; casos de uso opcionais (*optional use case*), que são necessários por apenas alguns membros da linha de produto; e casos de uso alternativos (*alternative use case*), onde diferentes casos de uso são necessários por vários membros da linha de produto. Foi introduzido o conceito de ponto de variação, de modo a modelar a variabilidade dentro do caso de uso. Estes pontos de variação podem ser modelados através das relações incluir (*Include*) e estender (*Extend*).

Para modelar casos de uso para linha de produtos, Gomaa introduziu o modelo de *features* para a abordagem UML. Baseando-se nas propriedades de reutilização de ambas abordagens, os casos de uso podem ser mapeados para as *features*. Uma *feature* pode corresponder a um único caso de uso, a um grupo de casos de uso, ou a um ponto de variação dentro de um caso de uso. Deste modo, os casos de uso e as *features* podem ser usados para complementar um ao outro. Existem várias maneiras que podem ser modeladas *features* em UML: modelação de *features* como um pacote de caso de uso, em que um grupo de *features* que podem ser reutilizadas; modelação de *features* como meta-classes, uma classe pode ser utilizada para

representar um elemento da modelação; e representações de *features* em tabela, em que as *features* podem ser representadas consistentemente em uma tabela de representação [35]. Uma vez que a abordagem trata os requisitos funcionais de uma linha de produtos de software, a proposta mostra que a modelação de casos de uso pode ser usada para representar linhas de produto de software.

4.2 I* aspectual e LPS

Em [36], é proposta o uso da abordagem i* aspectual para capturar as *features* comuns e variáveis em requisitos linha de produtos de software. De acordo com Silva et al [36], a selecção das *features* específicas e a sua decomposição para um produto individual com o núcleo de recursos da LPS é facilitada devido ao uso de princípio de orientação a aspectos. Combinar o i* aspectual com abordagens onde extraem mais cedo a variabilidade no ciclo de vida do desenvolvimento de uma LPS, enriquece a variabilidade capturada por i* aspectual. Esta variabilidade capturada em i* aspectual pode facilitar o inter-relacionamento entre vários tipos de requisitos (como por exemplo requisitos funcionais, não funcionais e organizacionais) em um mesmo modelo.

Para ilustrar como uma estratégia orientada a objectivo aspectual pode ser usada para representar a variabilidade em modelos de requisitos, os autores propuseram um conjunto de heurísticas para criar modelos i* aspectuais de modelos de *features*: A separação de *features* no modelo i*, os tipos de *features* e relacionamento nos modelos, verificação da correctude dos relacionamentos e o mapeamento dos nomes das *features* para os nomes dos elementos do modelo i* aspectual. Porém, a abordagem precisa de ser melhorada para reduzir a escalabilidade dos modelos, visto que com a adição dos aspectos acaba por ser necessário representa-los no modelo, dificultando a sua compreensão e legibilidade. A abordagem considera que cada *feature* opcional ou alternativa do modelo irá ser mapeada num aspecto, e isto nem sempre se verifica. De um modo geral, há pouca capacidade de gestão da complexidade do modelo, a representação de todas as variabilidades fica difícil a sua compreensão e existe falta de um mecanismo de modularização e composição [36].

4.3 Modelo de *features* e EROO

Em [37], os autores propuseram uma nova extensão orientada a modelos, para uma ferramenta de Engenharia de Requisitos Iniciais (*OpenOME*), que gera um modelo de *features* inicial de um sistema futuro para os objectivos dos *stakeholders*. Estes objectivos são identificados e analisados com a engenharia de requisitos orientada a objectivo mesmo antes dos requisitos

funcionais e não-funcionais de um sistema terem sido delineados. Os modelos de objectivos, como um resultado do processo de Identificação, são uma fonte natural de variabilidade intencional.

Segundo Yu et al., pela prática da programação geradora, os modelos de *features* representam a variabilidade do sistema e conduz à configuração de produtos finais. Na proposta é feita a descrição de como a ferramenta suporta a abordagem e como gere a rastreabilidade entre os objectivos e as *features*. Numa primeira fase é gerado o modelo de *feature* de um sistema, depois é feita uma ligação para *features* orientado ao sistema e é mostrado como eles são configurados usando algoritmos de racionalização do objectivo. Para esta proposta, uma vez que a ferramenta efectua gerações automáticas dos modelos, é muito fácil induzir o analista ao erro, visto que a sua confiança no modelo gerado pode implicar futuros problemas [37].

4.4 MATA e LPS

Em [38], é descrita uma abordagem para manutenção da separação de *feature* durante a modelação utilizando a linguagem de composição UML baseada em transformação de grafo, e uma abordagem para detecção de interacções estruturais indesejáveis entre modelos de *features* diferentes. De acordo com Jayaraman et al., as *features* base são expressas em termos de diagramas de classe do UML, diagramas de sequências e diagramas de estados e as *features* variáveis são especificadas em UMLT (*Unifield Modeling Language Transformation*), que é uma representação UML de transformações de grafos que indicam como é feita a modificação dos modelos base.

Deste modo, esta descrição corresponde a técnica MATA (*Modeling Aspects using a Transformation Approach*) que é utilizado no contexto de aspectos. As *features* são compostas automaticamente utilizando um mecanismo de reescrita de grafo e uma dupla análise crítica é usada para detectar interacções de *feature* estrutural. Estas interacções permitem verificar se existem inconsistências entre o diagrama de dependência de *features* e os modelos de *features* UML. Esta abordagem utiliza uma ferramenta para o seu suporte. Uma vez que a cada *feature* variável é modelada independentemente de outras, um utilizador pode em qualquer altura escolher um subconjunto de *features* disponíveis e gerar automaticamente um modelo para este conjunto de *features*. Assim, esta abordagem é um bom exemplo de aplicação da abordagem MATA em LPS [38].

4.5 Resumo

Nesta secção, uma lista de trabalhos relacionados foram apresentados. As abordagens descritas foram propostas por outros autores e permitem a modelação da variabilidade de uma linha de produto no desenvolvimento de um software.

No capítulo 5 é descrito a abordagem LPS-KAOS que define um conjunto de procedimentos para adaptar a abordagem KAOS para a especificação das linhas de produtos de software.

5 Abordagem LPS-KAOS

As linhas de produtos de software (LPSs) tiram partido da sua facilidade de reutilização, tendo vários benefícios, entre eles a diminuição do tempo de construção de um produto para o mercado, o aumento da satisfação do cliente, etc. Deste modo, como a abordagem KAOS não se encontra muito bem explorada para as LPSs, neste capítulo são definidos dois processos para ajudar no desenvolvimento das LPSs, a nível da Engenharia de Domínio e da Engenharia da Aplicação.

Na Engenharia de Domínio são definidas actividades para a obtenção dos objectivos de uma LPS usando a abordagem KAOS, de modo a ser possível identificar as *features* que irão fazer parte de um modelo de *features* para o sistema. Para além da identificação destes objectivos e *features* de um sistema, os modelos KAOS e de *features* que serão produzidos, vão permitir dar uma visão clara do sistema em termos de diagramas.

Uma vez especificados os modelos de uma LPS a nível de Domínio, estes serão utilizados a nível da Engenharia da Aplicação para a configuração de uma aplicação específica.

Para um melhor entendimento dos processos definidos, foi utilizado um exemplo prático de um sistema de parqueamento, para ilustrar a abordagem LPS-KAOS.

5.1 Definição do processo para a sua utilização

Para definição do processo para a utilização da abordagem de modo a especificar uma LPS foram definidos dois processos: um a nível da Engenharia de Domínio e outro a nível da Engenharia da Aplicação.

5.1.1 Engenharia de Domínio

Para a abordagem LPS-KAOS, na Engenharia de Domínio foi definido um processo como é ilustrado na Figura 5.1. Este processo é composto por um conjunto de actividades que se relacionam entre si, que permite a identificação das *features* de um sistema através do modelo de objectivos da abordagem KAOS, para a especificação das Linhas de Produto de Software.

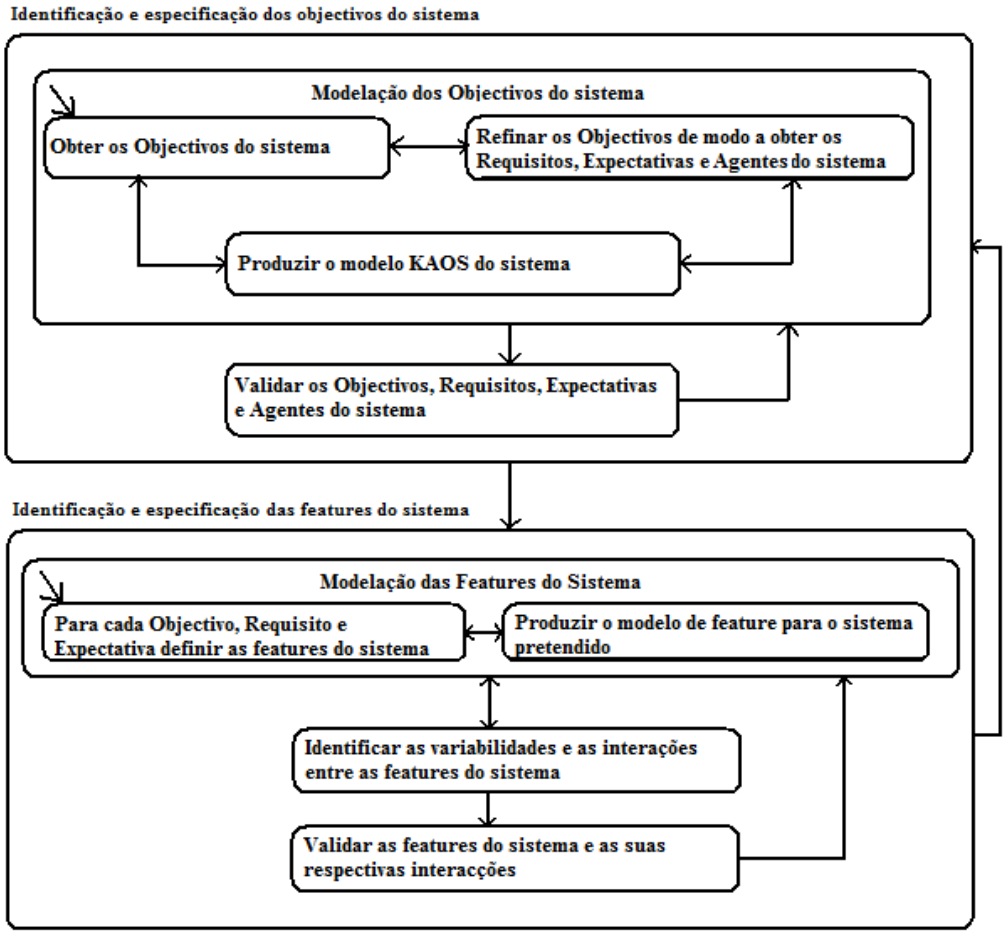


Figura 5.1 – Processo para a Engenharia de Domínio.

A actividade de identificação e especificação dos objectivos do sistema consiste na obtenção destes, na sua decomposição e no desenvolvimento do modelo de objectivos, de modo a facilitar a obtenção das *features* do sistema. Numa 1ª fase os objectivos são obtidos através de descrição do sistema, de entrevistas com os utilizadores, clientes e especialistas do domínio, da análise de sistemas existentes e outros métodos de descrição destes requisitos.

Uma vez obtidos os objectivos, estes podem ser decompostos em sub-objectivos que também podem dizer respeito a diferentes alternativas para atingir os mesmos, e inicia-se a construção

do modelo de objectivos. Os sub-objectivos são decompostos por refinamentos do tipo AND, e OR (para alternativas). Posteriormente são identificados os Requisitos e Expectativas para o modelo, de modo a torna-lo mais detalhado e perceptível para o que se pretende do sistema. Para além da decomposição dos objectivos do sistema, são também identificados os agentes que são responsáveis pelos requisitos e expectativas no sistema. Em seguida é efectuada a validação dos conceitos (objectivos, requisitos, expectativa ou agentes) do sistema através de técnicas de validação, tais como, a inspecção analítica, revisão profunda dos conceitos, prototipagem, outras [2]. Esta validação destina-se a mostrar que este modelo está de acordo com a especificação desejada e que atenda às expectativas dos clientes e utilizadores.

De notar que em certos casos (por exemplo: falta de um determinado requisito no modelo) na validação, há necessidade de voltar a fase de obtenção dos objectivos do sistema, de modo a solucionar o(s) problema(s) detectado(s). Por fim tem-se a produção do modelo de objectivos para o sistema.

Na identificação e especificação das *features* do sistema, são obtidas as *features*, as suas variabilidades e determinadas as interacções entre elas. Esta obtenção é realizada através dos conceitos definidos na actividade de identificação e especificação dos objectivos do sistema, isto é, para cada objectivo, requisito e expectativa do modelo de objectivos, será possível extrair uma ou mais *features* para o sistema. Na sub-actividade de modelação das *features* é efectuada esta consecução e em simultâneo é produzido o modelo de *feature* para o sistema pretendido. Nesta sub-actividade, para facilitar a decomposição das *features* em *sub-features* é efectuada a análise do modelo de objectivos do KAOS, isto é, ao analisar os refinamentos do modelo de objectivos poderemos elaborar a decomposição (por exemplo: *requires*, *xor*, etc.) das *features* presentes no modelo de *features*. Uma vez concluída a extracção das possíveis *features*, posteriormente identificam-se as variabilidades da LPS, isto é, distinguir quais *features* são obrigatórias e quais *features* são opcionais para a LPS. Outra propriedade importante a identificar é o relacionamento existente entre as diferentes *features*, isto é, identificar as restrições entre *features* de um sistema.

Após a especificação das *features*, é necessária a validação das decisões tomadas, de modo a que o sistema satisfaça as necessidades dos clientes e utilizadores, através das técnicas de validação descritas no processo de identificação e especificação dos objectivos do sistema, como por exemplo, revisão das *features* com especialistas do domínio e com os clientes de modo a saber se o sistema foi modelado de acordo com as suas especificações desejadas.

Nesta actividade de identificação e especificação das *features* do sistema, é também possível rever as decisões tomadas em caso de detecção de situações improváveis (como por exemplo: falta de *features* no modelo de *features*) na validação, permitindo voltar a sub-actividade anterior para solucionar as situações.

5.1.2 Engenharia de Aplicação

Na Engenharia de Aplicação foi definido uma actividade que se encontra decomposto como é ilustrado na Figura 5.2.

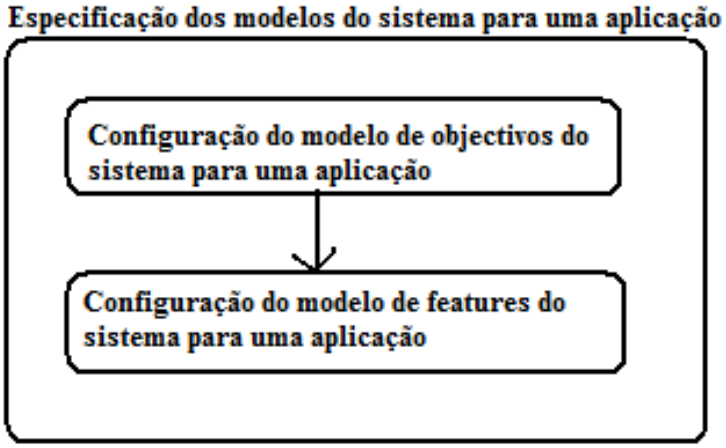


Figura 5.2 - Processo para a Engenharia de Aplicação.

A nível da Engenharia da Aplicação, os modelos de objectivos e de *features* produzidos durante a Engenharia de Domínio são analisados e configurados para uma determinada instância da LPS. Tanto para o modelo de objectivos como o de *features* são configurados para o produto pretendido.

Na secção seguinte será apresentado um exemplo prático para ilustrar o processo definido a nível da Engenharia de Domínio e da Engenharia da Aplicação.

5.2 Aplicação da abordagem LPS-KAOS

Para ilustrar as actividades dos dois processos da abordagem, será utilizada uma versão muito simplificada do sistema de parqueamento de automóveis, em que, para além da variante de entrada e saída do parque (utilização de um identificador de acesso ao parque) já estudada até agora, serão acrescentadas mais duas variantes, cujos requisitos são estabelecidos da seguinte forma [23]:

“Para que se possa ter acesso a um sistema de parqueamento, um cliente tem de obter um bilhete de uma máquina depois de pressionar um botão. Posteriormente, é permitida a entrada ao automóvel e estacionar em um lugar disponível. O sistema tem de controlar se o parque se encontra completo ou se existem lugares disponíveis e se encontra fechado ou não. Quando o cliente pretender abandonar o parque, ele tem de efectuar o pagamento numa das máquinas para este efeito, através do bilhete obtido na máquina de entrada consoante o tempo de estadia no parque. Depois que o cliente tiver pago, pode abandonar o parque inserindo o bilhete numa máquina de saída, que posteriormente será aberta a cancela.

Para além da entrada no parque através de bilhete, iremos ter em conta a variante de parqueamento até agora estudada, em que o sistema deve permitir o uso de um identificador como usado na Via-Verde para aceder ao parque. Relembrando os requisitos relacionados com o identificador, este pode ser obtido através de um simples processo de adesão, onde o cliente deve fornecer os seus dados pessoais, do seu cartão de débito e ainda do veículo a registar. É necessário fazer a activação do identificador numa caixa de multibanco associando este ao cartão de débito.

Para aceder a um parque basta que o identificador seja colocado no pára-brisas do veículo e aproximá-lo de uma das cancelas especiais, que se abrirá se ele for válido e acenderá uma luz verde. Para sair do parque o procedimento é semelhante. A quantia a pagar depende do tempo gasto no parque e é mostrada num visor acoplado a uma cancela de saída.

Outra variante para o parqueamento é a utilização de um cartão tipo “Via-Card”. O processo de adesão para obtenção do Cartão e semelhante ao do Identificador. Para este caso, o cliente aproxima o cartão ao leitor que se encontra na entrada e no caso de o cartão ser válido, permite o acesso ao parque. Caso contrário o cliente deve retirar um bilhete que permite a sua entrada no parque. Para a saída, o procedimento é o mesmo, e é mostrado num visor o valor a pagar.”

Nas subsecções seguintes é discutido o problema tanto a nível da Engenharia de Domínio como da Engenharia da Aplicação.

5.2.1 A nível de Engenharia de Domínio

As actividades da Engenharia de Domínio serão descritas nesta subsecção de acordo com o processo descrito na secção 5.1.

5.2.1.1 Identificação e especificação os objectivos do sistema

Esta Actividade é decomposta em duas sub-actividades: Modelação dos Objectivos do sistema e Validar os Objectivos, Requisitos, Expectativas e Agentes do sistema. De um modo geral, são definidos os objectivos do sistema a desenvolver e a construção do modelo de objectivos para este sistema. Em primeiro lugar são obtidos os objectivos de alto nível que posteriormente serão mais detalhados, isto é, decompostos em níveis mais baixos de abstracções. Depois da produção do modelo de objectivos completo, ilustrando graficamente os objectivos deste sistema, o processo é consolidado através da validação das decisões tomadas.

i. Modelação dos objectivos do sistema

Na fase inicial do desenvolvimento de todo e qualquer sistema, é importante conhecer os objectivos destes, isto é, é necessário conhecer como o sistema deve comporta-se, que restrições subsistem, e que outras características do sistema são importantes. Deste modo, como na abordagem KAOS os requisitos de sistema são obtidos em forma de objectivos e através da descrição de requisitos, temos identificado alguns objectivos deste sistema:

- Efectuar adesão ao sistema;
- Efectuar a activação (Cartão/Identificador) no sistema;
- Efectuar a entrada no parque;
- Efectuar a saída do parque.

Alguns destes objectivos podem ser atingidos de forma diferente, isto é, eles podem ser decompostos em vários sub-objectivos alternativos que representam as várias opções de se alcançarem estes objectivos. Estas opções serão referenciadas com refinamentos do tipo *AND* da abordagem KAOS. Na Tabela 5.1, é ilustrado como estes objectivos podem ser alcançados.

Tabela 5.1 - Objectivos e Sub-Objectivos (Alternativas)

Objectivos	Sub-Objectivos (alternativas)	Descrição
- Efectuar a adesão ao sistema	1) Efectuar a adesão ao identificador; 2) Efectuar a adesão ao cartão.	- O cliente pode optar por aderir a um Identificador ou a um Cartão para ter acesso ao parque.
- Efectuar a activação (Cartão/Identificador) no sistema	1) Efectuar a activação do identificador; 2) Efectuar a activação do cartão.	- Uma vez que a activação pode ser efectuada de duas formas possíveis é necessária a sua distinção.

- Efectuar a entrada no parque	1) Efectuar a entrada no parque através do bilhete; 2) Efectuar a entrada no parque através do cartão; 3) Efectuar a entrada no parque através do identificador.	- Existe 3 formas possíveis de se efectuar o parqueamento. Deste modo, é importante distinguir as várias alternativas disponíveis para o efeito.
- Efectuar a saída do parque	1) Efectuar a saída do parque por bilhete; 2) Efectuar a saída do parque por cartão; 3) Efectuar a saída do parque por identificador.	- No caso da saída do parque, é necessário ter em conta a uma das 3 formas de entrada, isto é, a saída é efectuada de acordo com forma de entrada no parque (Cartão, Identificador ou bilhete).

Após a identificação dos objectivos desejados do sistema, começa-se a especificação do modelo de objectivos do sistema de modo a ter uma visão diagramática do mesmo e contribuindo para a consolidação do raciocínio para o sistema pretendido.

Na Figura 5.3 é ilustrado o modelo de objectivos principais do sistema de parqueamento que descreve os objectivos identificados na Tabela 5.1.

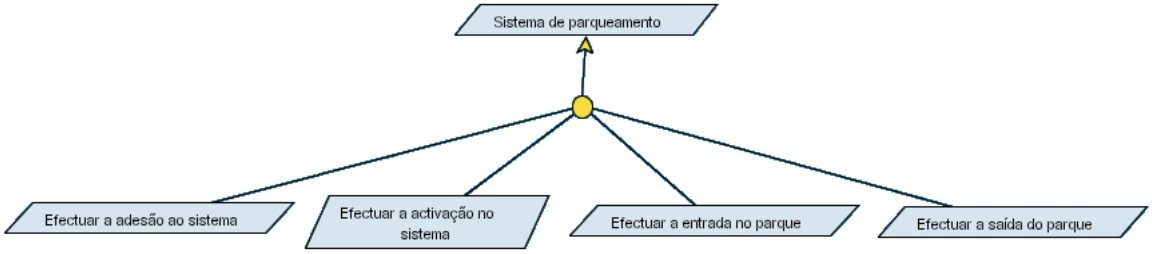


Figura 5.3 - Sistema de Parqueamento (objectivos).

Para atingir os objectivos descritos do sistema na abordagem KAOS é necessário decompô-los (refinamentos do tipo OR ou AND) em sub-objectivos, requisitos e/ou expectativas. Por sua vez, estes sub-elementos encontram-se num nível de abstracção mais baixo em relação aos objectivos principais do sistema. Isto permite realçar as acções que são realizadas para alcançar um determinado objectivo.

Deste modo, para o caso do objectivo "Efectuar adesão ao sistema" que subdivide-se (alternativas) em "Efectuar adesão ao cartão" e "Efectuar a adesão ao identificador" é possível identificar algumas expectativas, visto que a adesão dos serviços depende apenas do fornecimento dos dados necessários para o efeito por parte do cliente e um requisito que permite guardar os dados pessoais no sistema. Como estes dados são fornecidos pelo cliente e a gravação destes são efectuados por um sistema de controlo, temos identificado os seguintes agentes do sistema: o cliente e o sistema de controlo de parque.

Para o objectivo “Efectuar a activação no sistema”, uma vez que se efectua a activação do “Cartão” e do “Identificador” (alternativas) por parte do cliente, existirão expectativas para o efeito. O agente cliente já identificado será também responsável por estas expectativas.

Na Tabela 5.2 são ilustradas as expectativas e requisitos dos objectivos “Efectuar adesão ao sistema” e “Efectuar activação no sistema”.

Tabela 5.2 - Expectativas dos objectivos "Efectuar Adesão ao sistema" e “Efectuar Activação no sistema”

	Efectuar adesão ao cartão e Efectuar adesão ao identificador	**Efectuar activação do identificador e Efectuar activação do cartão**
Expectativas	-Fornecer dados pessoais; -Fornecer dados do veículo; -Fornecer dados do cartão de débito.	-Associar cartão de débito ao identificador; -Associar cartão de débito ao cartão.
Requisitos	-Guardar os dados fornecidos	

Uma vez identificados os requisitos, expectativas e os agentes para os objectivos citados, continua-se a desenvolver o modelo de objectivos para uma melhor descrição destes. Deste modo, a Figura 5.4 ilustra o objectivo “Efectuar a adesão ao sistema” que descreve a sua decomposição em vários sub-objectivos alternativos e as suas respectivas expectativas, requisitos e agentes do sistema.

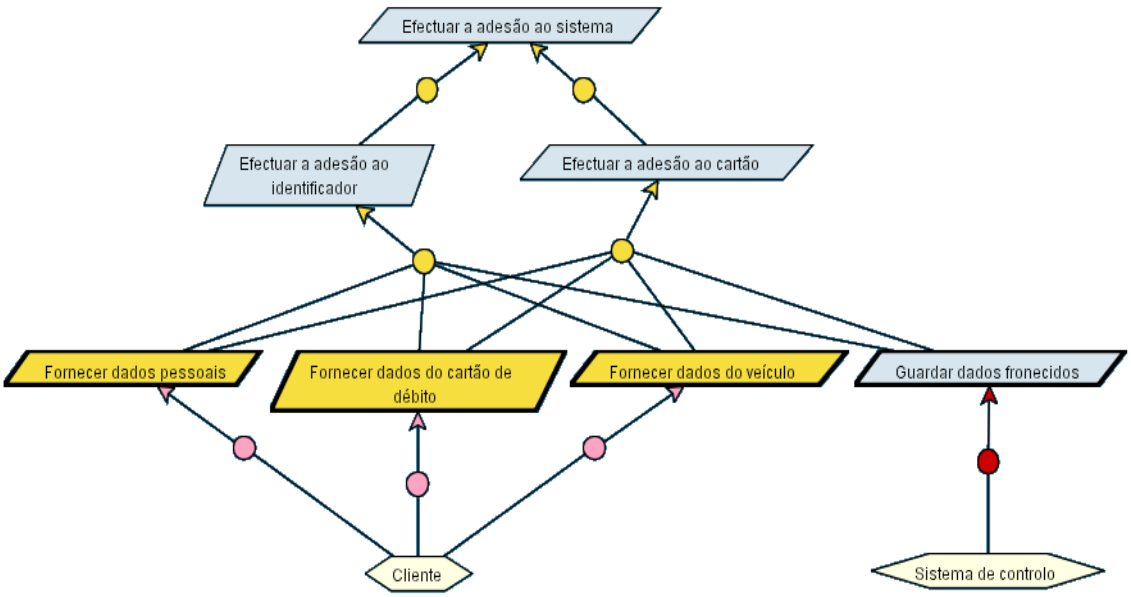


Figura 5.4 - Objectivo "Efectuar a adesão ao sistema" e sua decomposição.

A Figura 5.5 descreve o objectivo “Efectuar a activação no sistema” demonstrando a sua decomposição e as suas respectivas propriedades.

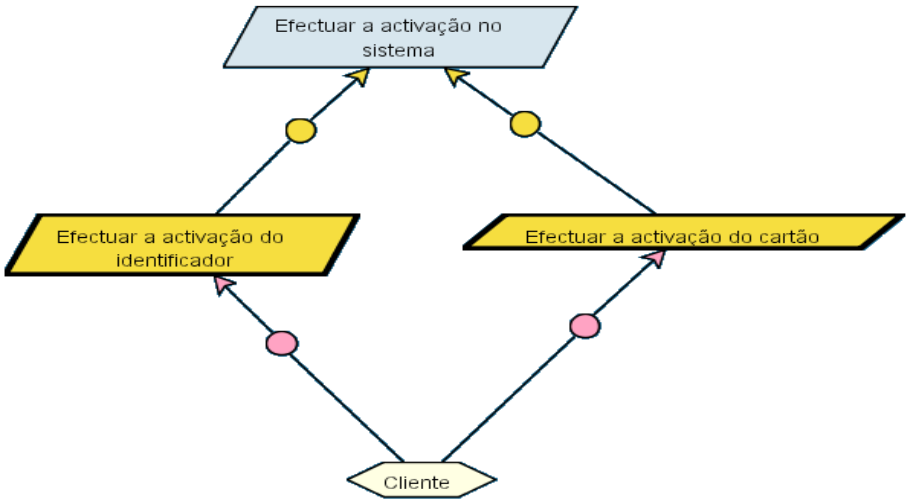


Figura 5.5 - Objectivo "Efectuar a activação no sistema" e a sua decomposição.

Para os objectivos “Efectuar entrada no parque” e “Efectuar saída no parque” é possível identificar vários requisitos e expectativas que fazem parte do sistema. Uma vez que estes objectivos podem ser atingidos de três formas diferentes (através do bilhete, do identificador ou do cartão), é importante realçar que para cada uma destas maneiras de atingir os objectivos, tem-se presente conjunto de requisitos e expectativas.

Na Figura 5.6 e na Figura 5.7 são ilustradas a decomposição dos objectivos “Efectuar a entrada no parque” e “Efectuar a saída do parque” respectivamente.

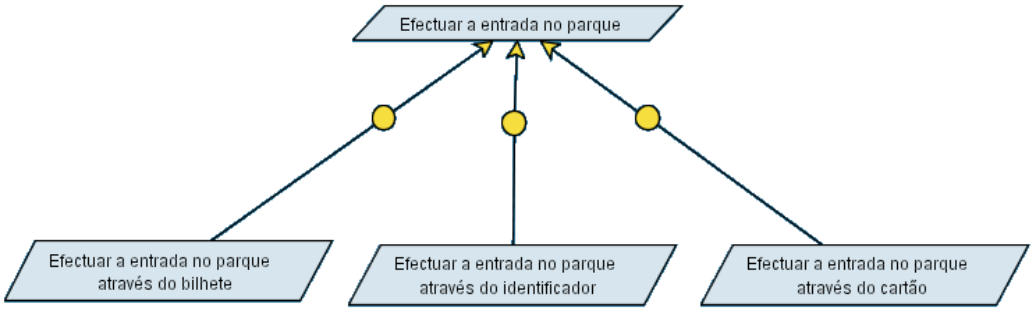


Figura 5.6 - Objectivo "Efectuar a entrada no parque" e sua decomposição.

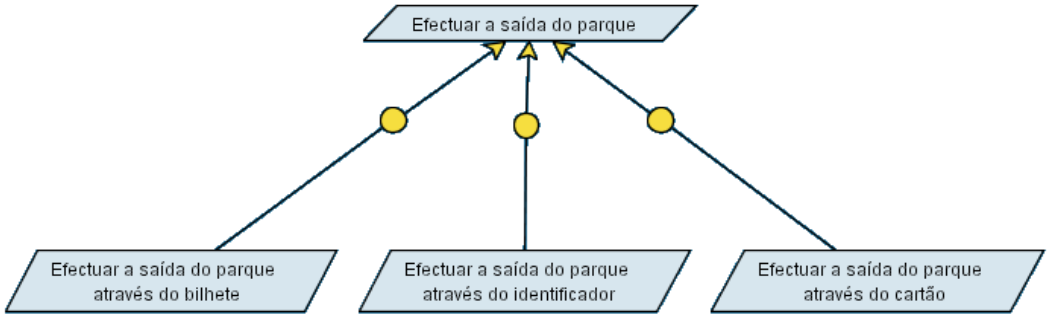


Figura 5.7 - Objectivo "Efectuar a saída do parque" e sua decomposição.

Após a identificação das três alternativas diferentes de atingir os objectivos “Efectuar a entrada no parque” e “Efectuar a saída do parque”, seguidamente é descrito detalhadamente cada alternativa de modo a atingir os objectivos em causa. Estas formas de entrada e saída de um parque de estacionamento apresentam algumas diferenças no seu tratamento.

Na Tabela 5.3, são ilustrados vários requisitos e expectativas que compõem o sub-objectivo “Efectuar a entrada no parque através do bilhete”.

Tabela 5.3 - Requisitos e Expectativas do objectivo "Efectuar a entrada no parque através do bilhete".

	Efectuar a entrada no parque através do bilhete	**Descrição**
Expectativas	-Pressionar o botão da máquina de entrada; -Retirar o Bilhete na máquina de entrada.	-Para a entrada no parque o cliente precisa de obter o bilhete, pressionando um botão e retirar o bilhete na máquina de entrada.
Requisitos	-Acender a luz do semáforo; -Guardar entrada; -Abrir a cancela.	-Após a obtenção do bilhete na máquina de entrada por parte do cliente, o sistema realiza um conjunto de acções de modo a permitir a entrada do veículo.

Na Figura 5.8 é demonstrada a representação gráfica da decomposição do objectivo “Efectuar a entrada no parque através do bilhete”.

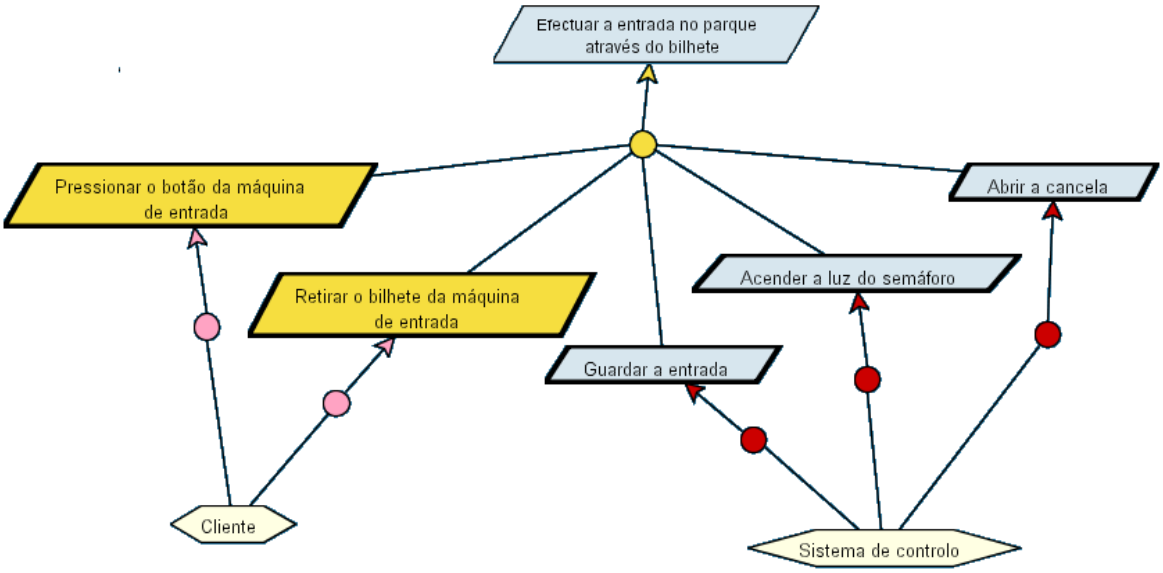


Figura 5.8 - Objectivo "Efectuar a entrada no parque através do bilhete" e sua decomposição.

Para o sub-objectivo “Efectuar a saída do parque através do bilhete” para além dos requisitos expectativas a qual é constituído existe um sub-objectivo “Efectuar pagamento do bilhete na

máquina de pagamento" que é necessário atingir para satisfazer o objectivo de alto nível. Este sub-objectivo é composto por alguns requisitos e expectativas de modo a ser satisfeito.

Na Tabela 5.4 são descritos os requisitos e expectativas que fazem parte do sub-objectivo "Efectuar o pagamento do bilhete na máquina de pagamento".

Tabela 5.4 - Requisitos e Expectativas do sub-objectivo "Efectuar o pagamento do bilhete na máquina de pagamento".

	Efectuar o pagamento do bilhete na máquina de pagamento	**Descrição**
Expectativas	-Inserir o bilhete na máquina de pagamento; -Inserir o dinheiro; -Retirar o Bilhete.	-Para se efectuar o pagamento na máquina de pagamento, o cliente precisa de cumprir certas acções para sair do parque.
Requisitos	-Calcular o valor a pagar; -Mostrar o valor a pagar no visor; -Devolver troco.	-Após a inserção do bilhete na máquina e inserção da quantia adequada, o sistema vai realizar um conjunto de acções para atingir o objectivo.

Os requisitos e expectativas que permitem alcançar o objectivo "Efectuar a saída do parque através do bilhete" encontram-se representados na Tabela 5.5.

Tabela 5.5 - Requisitos e Expectativas do objectivo "Efectuar a saída do parque através do bilhete".

	Efectuar a saída do parque através do bilhete	**Descrição**
Expectativas	-Inserir o bilhete na máquina de saída.	-Após o pagamento da estadia no parque, o cliente desloca-se a saída para proceder a saída.
Requisitos	-Verificar o bilhete -Acender a luz do semáforo; -Abrir a cancela.	-Uma vez inserido o Bilhete na máquina de saída, o sistema realiza um conjunto de acções para concluir a saída do cliente.

Depois de mais uma vez identificar alguns objectivos, requisitos, expectativas e possíveis agentes do sistema, os mesmos são representados graficamente no modelo de objectivos na Figura 5.9.

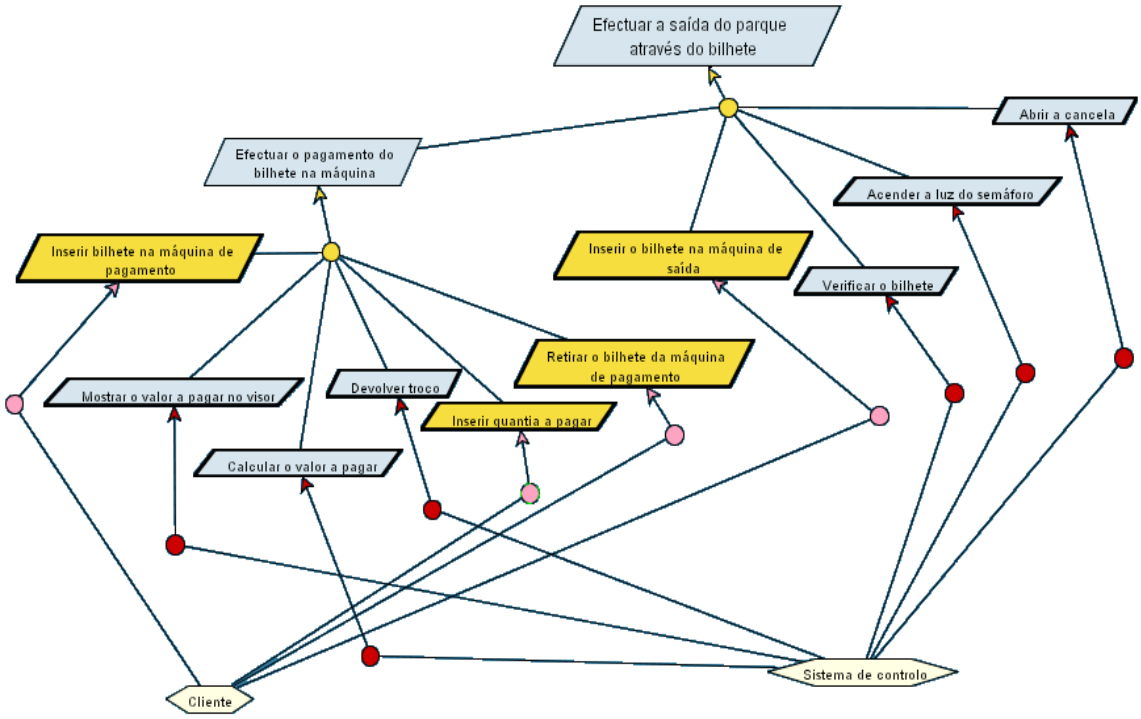


Figura 5.9 - Objectivo "Efectuar a saída do parque através do bilhete" e sua decomposição.

Outra possível forma alternativa de se atingir os objectivos “Efectuar a entrada no parque” e “Efectuar a saída do parque” é através de um identificador destinado para tal.

Na Tabela 5.6 é ilustrado um conjunto de requisitos e expectativas do objectivo “Efectuar a entrada no parque através do identificador”.

Tabela 5.6 - Requisitos e Expectativas do objectivo "Efectuar a entrada no parque através do identificador".

	Efectuar a entrada no parque através do identificador	**Descrição**
Expectativas	-Dirigir-se a cancela;	-Para este caso é necessária a aproximação do veículo a zona da Cancela para que o Identificador possa ser detectado.
Requisitos	-Detectar identificador através do sensor; -Validar identificador; -Acender a luz do semáforo; -Registar hora de entrada; -Abrir a Cancela.	-Após a aproximação do veículo a entrada, o sistema realiza um conjunto de acções de modo a permitir a entrada do mesmo.

A Figura 5.10 ilustra o tratamento da entrada do parqueamento utilizando o identificador, isto é, a entrada dos veículos no parque através do identificador.

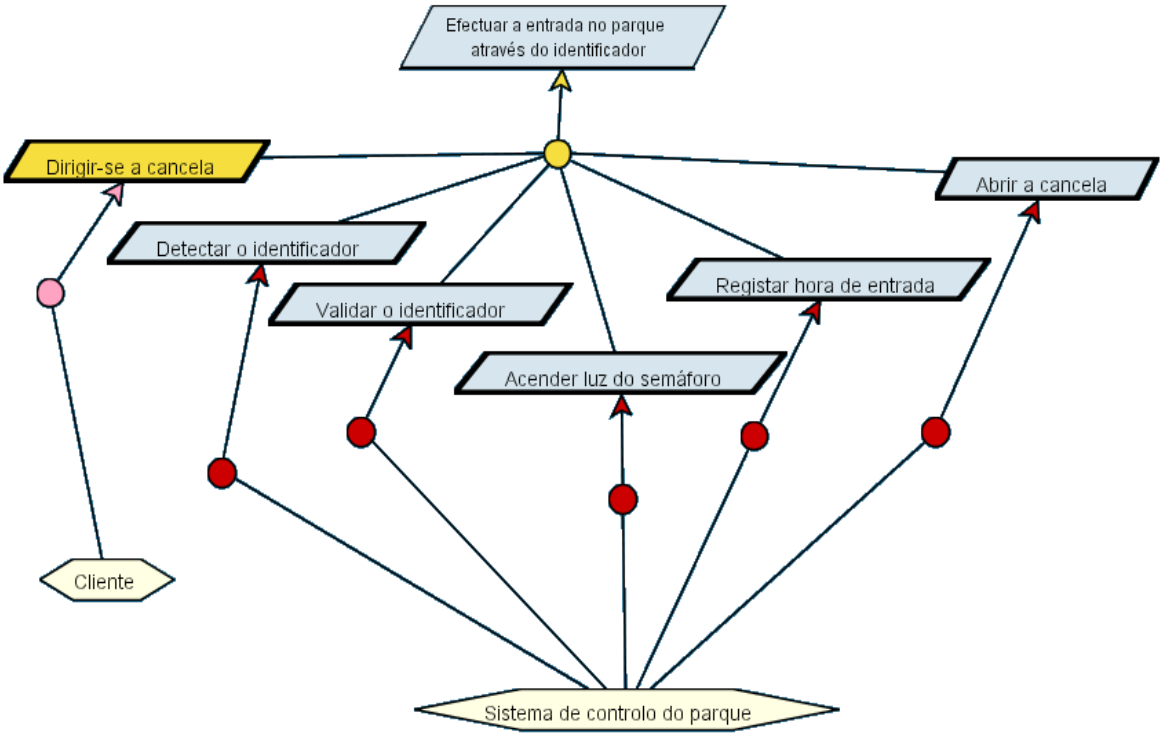


Figura 5.10 - Objectivo "Efectuar a entrada no parque através do identificador" e sua decomposição.

Para "Efectuar a saída do parque através do identificador" podem-se verificar alguns requisitos e expectativa comuns ao objectivo "Efectuar a entrada no parque através do identificador". Estes e outros requisitos encontram-se descritos na Tabela 5.7.

Tabela 5.7 - Requisitos e Expectativas do objectivo "Efectuar a saída do parque através do identificador".

	Efectuar a saída do parque através do identificador	**Descrição**
Expectativas	-Dirigir-se a cancela;	-Para este caso é necessária a aproximação do veículo a zona da Cancela para que o Identificador possa ser detectado.
Requisitos	-Detectar o identificador através do sensor; -Validar o Identificador; -Registar hora de saída; -Calcula valor a pagar; -Mostrar o valor a pagar no visor -Acender a luz do semáforo; -Abrir a cancela.	-Após a aproximação do veículo a saída, o sistema realiza um conjunto de acções de modo a permitir a saída do mesmo.

Estes requisitos, expectativas e agentes descritos para o objectivo "Efectuar a saída do parque através do identificador" encontram-se ilustrados na Figura 5.11.

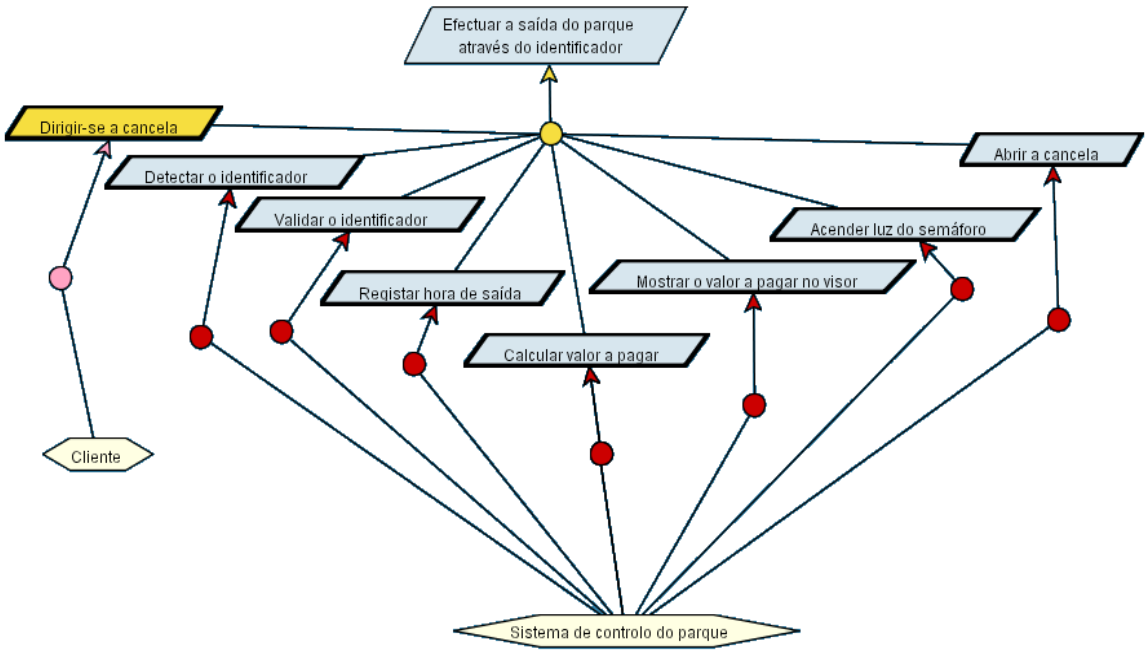


Figura 5.11 - Objectivo "Efectuar a saída do parque através do identificador" e sua decomposição.

Para os dois sub-objectivos "Efectuar entrada no parque através do cartão" e "Efectuar saída do parque através do cartão", é possível identificar também alguns requisitos e expectativas que fazem parte do sistema. Na Tabela 5.8 são descritos os mesmos para o caso de "Efectuar entrada no parque através do cartão".

Tabela 5.8 - Requisitos e Expectativa do objectivo "Efectuar a entrada no parque através do cartão".

	Efectuar a entrada no parque através do cartão	**Descrição**
Expectativas	-Passar o cartão no leitor	-Para a entrada no parque o cliente precisa de passar o dispositivo (cartão) próximo ao leitor de cartões para se processar a entrada.
Requisitos	-Ler o cartão -Validar o Cartão; -Acender a luz do semáforo; -Registar hora de entrada; -Abrir a cancela.	-Após a detecção do cartão por parte de um leitor de cartões, o sistema realiza um conjunto de acções de modo a permitir a entrada do veículo.

Para concluir as variantes do sistema de parqueamento, a Figura 5.12 mostra o modelo de entrada para o caso do Cartão. Nesta figura é demonstrada a decomposição do objectivo em causa.

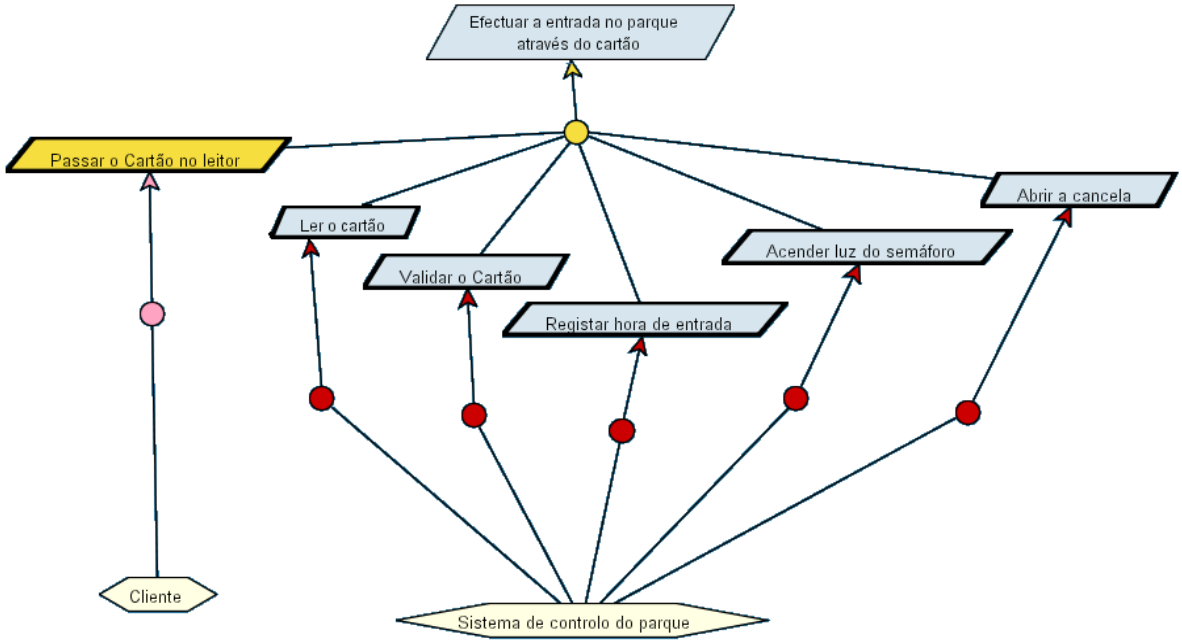


Figura 5.12 - Objectivo "Efectuar a entrada no parque através do cartão" e sua decomposição.

Na Tabela 5.9 são ilustrados os requisitos e expectativas que fazem parte do caso de "Efectuar a saída do parque através do cartão". Neste objectivo existe também alguns requisitos e expectativas que são comuns ao objectivo "Efectuar a entrada no parque através do cartão", visto que algumas acções desenroladas na entrada com o cartão também encontram-se presentes na saída com o cartão.

Tabela 5.9 - Requisitos e Expectativa do objectivo "Efectuar a saída do parque através do cartão".

	Efectuar a saída do parque através do cartão	**Descrição**
Expectativas	-Passar o cartão no leitor	- Para a saída no parque o cliente precisa apenas de passar o dispositivo (cartão) próximo ao leitor de cartões para se processar a saída.
Requisitos	-Ler o cartão; -Validar cartão; -Registar hora de Saída; -Calcula valor a pagar; -Mostrar o valor a pagar no Visor -Acender a luz do semáforo; -Abrir a cancela.	-Após a detecção do cartão, o sistema realiza um conjunto de acções de modo a permitir a saída do veículo.

A Figura 5.13 ilustra o objectivo "Efectuar a saída do parque através do cartão" e sua respectiva decomposição.

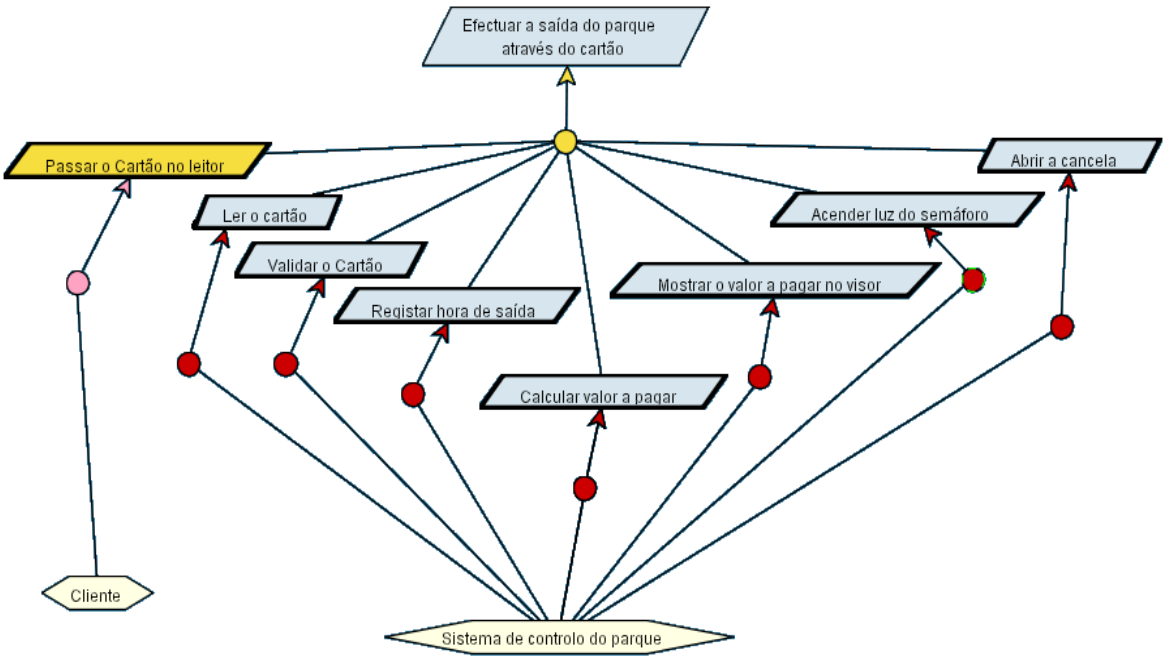


Figura 5.13 - Objectivo "Efectuar a saída do parque através do cartão" e sua decomposição.

Depois da identificação dos conceitos e do desenvolvimento do modelo de objectivos que irão fazer parte do sistema, é necessário efectuar a validação dos mesmos. Esta é efectuada na subsecção seguinte através da sub-actividade de validar os Objectivos, Requisitos, Expectativas e Agentes do sistema.

ii. Validar os Objectivos, Requisitos, Expectativas e Agentes do sistema

Uma vez identificados os possíveis objectivos, requisitos e expectativas do sistema é necessário efectuarem a validação dos mesmos. Esta validação destina-se a mostrar que estas propriedades estão de acordo com a especificação e que atendem às expectativas dos clientes e utilizadores.

A validação visa a assegurar se um determinado sistema realiza aquilo que o cliente deseja. Existem diferentes formas de validação, de entre elas a inspecção analítica, revisão de modelos, prototipagem, outras. Entretanto não é objectivo deste trabalho detalhar esta actividade, apenas vamos assumir que foi realizada de modo a garantir validade do sistema desenvolvido.

5.2.1.2 Identificação e especificação das *features* do sistema

Após a identificação e especificação dos objectivos do sistema, na actividade seguinte são identificadas as *features* que irão fazer parte do sistema, visto que, o que se pretende é adaptar

a abordagem KAOS para especificar a LPS. Deste modo, estas *features* serão obtidas através dos Objectivos, Requisitos e Expectativas já definidos na actividade anterior.

i. Modelação das *features* do sistema

Esta actividade baseia-se em duas sub-actividades importantes: Para cada conceito (objectivos, requisitos e expectativas) definir as *features* do sistema e produzir o modelo de *features* para a LPS pretendida. O tratamento destes processos é efectuado em simultâneo, isto porque facilita o desenvolvimento de uma LPS, tornando mais flexível a sua compreensão.

Uma vez consolidados os objectivos da LPS, é importante realçar que estes são indispensáveis para a identificação das *features* da mesma. Esta identificação baseia-se na descrição dos objectivos, requisitos e expectativas do sistema, compondo as seguintes heurísticas (H) para a obtenção das *features*:

1. Identificação das *features* através dos nomes próprios nos conceitos.
   - Para cada conceito identificar as propriedades centrais destes (por exemplo, para Validar Identificador temos Identificador);
   - Elaborar uma lista de todas as propriedades extraídas dos conceitos;
   - Cada propriedade poderá corresponder a uma *feature* do sistema no modelo de *feature* (Exemplo: Identificador).

2. Identificação das *features* do sistema através dos verbos que têm origem nas actividades do sistema. Através dos verbos presentes nos conceitos definidos e que dão a origem a certas funcionalidades do sistema, é possível a identificação de *features* para o sistema. Por exemplo, o objectivo "Detectar identificador" ilustrado no modelo de objectivos das Figura 5.10 e Figura **5.11** permite-nos obter uma *feature* denominada "Detecção do ID" que dá a origem a actividade de entrada de um veículo através do identificador.

3. Identificação das *features* do sistema através de recursos requeridos por outras *features*. Por exemplo, para o objectivo "Detectar o identificador" é possível obter a *feature* "Identificador" que requer o recurso "sensor" para a sua detecção. Deste modo, é também identificada uma *feature* requerida denominada "Sensor".

4. Identificação da *feature* conceptual do sistema. O objectivo de alto nível que representa o sistema no modelo de objectivo da abordagem KAOS, origina a *feature* conceitual para o modelo de *features*. Como por exemplo "Sistema de parqueamento" suscita a existência de uma *feature* "Parqueamento". É útil para a definição da hierarquia de *features*.

Uma vez consolidada a identificação das possíveis *features* do sistema, estas são adicionadas ao glossário do domínio associando a cada uma, uma pequena descrição.

Deste modo, na Tabela 5.10 são descritos os objectivos e sub-objectivos (de notar que os requisitos e expectativas ainda não se encontram tratados) do sistema de parqueamento e possíveis *features* identificadas através dos mesmos. Para a construção desta tabela foram utilizadas os modelos de objectivos das Figuras Figura 5.3, Figura 5.4, Figura 5.6 e Figura 5.7 e as heurísticas de definidas no parágrafo anterior. Nesta tabela verifica-se que objectivos diferentes podem estar relacionados às mesmas *features*, permitindo reforçar a presença das mesmas no sistema a desenvolver.

Tabela 5.10 - Identificação de *features* a partir dos objectivos do Sistema.

Objectivos	***Features***	**Descrição**	**H**
-Sistema de parqueamento	-Parqueamento	-Feature "Parqueamento" que representa o sistema em desenvolvimento.	4
-Efectuar a adesão ao sistema	-Adesão_Sistema	-*Feature* para o caso da adesão dos clientes.	1
-Efectuar a adesão ao identificador	-Identificar -Adesão_Sistema (requerida)	-Uma vez que o identificador é uma das variantes para se aceder o parque, logo é uma propriedade importante para o sistema que requer a sua adesão.	1, 3
-Efectuar a adesão ao cartão	-Cartão -Adesão_Sistema (requerida)	-Uma vez que o cartão é uma das variantes para se aceder o parque, logo é uma propriedade importante para o sistema que requer a sua adesão.	1, 3
-Efectuar a activação no sistema	-Activação	-*Feature* para a validação ou activação dos dispositivos.	1
-Efectuar a entrada no parque	-Maq_Entrada	-Como o parque requer pelo menos uma entrada (Maq_Entrada) e uma saída é necessário efectuar a distinção entre estas duas características, já que elas podem processar de modo diferente.	1
-Efectuar a entrada através do bilhete	-Bilhete -Maq_Ent_B (requerida)	-Uma vez que o bilhete é uma das variantes para se aceder o parque, logo é uma propriedade importante para o sistema para aceder ao parque através de uma máquina de entrada com bilhetes.	1,3
-Efectuar a entrada através do identificador	-Identificador -Maq_Ent_ID (requerida)	-*Feature* "Identificador" já identificada e requer uma máquina de entrada com o identificador para aceder ao parque.	1,3
-Efectuar a entrada através do cartão	-Cartão -Maq_Ent_C (requerida)	-*Feature* "Cartão" já identificada e requer uma máquina de entrada com o cartão para aceder ao parque.	1,3
-Efectuar a saída do Parque	-Maq_Saída	-Como o parque requer pelo menos uma entrada e uma saída (Maq_Saída) é necessário efectuar a distinção entre estas duas características, já que elas podem ser processadas de modo diferente.	1
-Efectuar o pagamento	-Maq_Pag_B	-Para o caso do bilhete é necessário a	1,3

do bilhete	-Bilhete (requerida)	identificação de uma máquina de pagamento para efectuar o pagamento no caso do bilhete.	
-Efectuar a saída através do Bilhete	-Maq_Saída_B -Bilhete (requerida)	-*Feature* "Bilhete" já identificada, requerida na máquina de saída com o bilhete.	1,3
-Efectuar a saída através do identificador	-Maq_Saída_ID -Identificador (requerida)	-*Feature* "Identificador" já identificada, requerida na máquina de saída com o identificador.	1,3
-Efectuar a saída através do cartão	-Maq_Saída_C -Cartão (requerida)	-*Feature* "Cartão" já identificada, requerida na máquina de saída com o cartão.	1,3

Depois de identificadas algumas *features* do sistema, começa-se a produzir o modelo de *features*, permitindo ter uma visão mais clara da LPS que se pretende. Para o desenvolvimento do modelo de *features* do sistema, temos as seguintes heurísticas:

A. Analisando os refinamentos do modelo de objectivos da abordagem KAOS, vai permitir ajudar na decomposição das *features* que irão fazer parte do modelo de *features*. Alguns relacionamentos entre as *features* dos modelos de *features* vão ser justificados através dos refinamentos dos objectivos do modelo de objectivos que dão a origem a estas *features*. Como por exemplo, o objectivo "Efectuar a entrada no parque", que dá origem a *feature* "Maq_Entrada", é decomposto pelos sub-objectivos "Efectuar a entrada no parque através do bilhete", "Efectuar a entrada no parque através do identificador" e "Efectuar a entrada no parque através do cartão", dando origem as *features* "Maq_Ent_B", "Maq_Ent_ID" e "Maq_Ent_C" respectivamente. Com a ajuda do modelo de objectivos, a "Maq_Entrada" será decomposta pelas *features* produzidas pelos seus sub-objectivos no modelo de *features* a desenvolver.

B. Através das alternativas do modelo de objectivos do sistema são definidas as variabilidades que irão fazer parte do modelo de *features* deste sistema. As alternativas dos modelos de objectivos vão permitir a identificação das ligações do tipo *OR* ou *XOR* do modelo de *features*. Para ajudar nesta decisão, recorre-se ao documento de descrição do sistema, de modo a saber que tipo de ligação se refere. Como por exemplo, as diferentes formas de aceder o parque de estacionamento através das máquinas de entrada (bilhete, identificador e cartão), permite identificar as ligações do tipo "OR" no modelo de *features* entre os elementos, garantido a alguma variabilidade deste.

C. Os documentos que descrevem o sistema para o seu desenvolvimento e o modelo de objectivos, vão permitir identificar as *features* obrigatórias e opcionais para um modelo de *features*. Estes documentos fornecem informações que numa 1ª vista consegue-se extrair propriedades (por ex: propriedades comuns e variáveis) para a modelação das

features. Através dos modelos de objectivos é possível saber que conceitos são utilizados com maior ou menor frequência, permitido identificar as variabilidades das *features* obtidas através destes conceitos. Por exemplo, a descrição do pagamento da estadia no parque através do bilhete é referente ao caso do bilhete, permitindo apenas considerar que em caso da existência de uma *feature* (Maq_Pag_B) para o efeito no parque, será para o caso bilhete, excluindo as outras variantes de parqueamento. Deste modo a *feature* identificada para este efeito será "opcional" não fazendo parte de toda a configuração possível do sistema produzido.

D. Quando existir um grupo de *features* com ligações do tipo *OR* ou *XOR* pode ser necessário a criação de uma *feature* suplementar que represente este grupo de modo a estruturar o modelo. A decomposição de uma feature em várias *sub-features*, em que nestas features *sub-features* existam grupos de features com ligações do tipo *OR* ou *XOR*, é possível a criação de uma *feature* suplementar de modo a integrar estes grupos de *features*, permitindo dar uma melhor visão destes agrupamentos no modelo de *features*. Por exemplo, se a *feature* Maq_Entrada tivesse outras *features* além do grupo OR constituído por "Maq_Ent_B", "Maq_Ent_ID" e "Maq_Ent_C", estas poderiam ser agrupadas em uma *feature* suplementar (como por exemplo: Tipo_Maq) e esta seria uma *sub-feature* de "Maq_Entrada".

Para a LPS em desenvolvimento algumas variabilidades do mesmo podem já ser identificadas na fase de identificação das *features* de modo a ser consolidadas na sub-actividade seguinte, que permite identificar as variabilidades e interacções entre as *features* do sistema. Algumas variabilidades por identificar serão apresentadas à partida como obrigatórias, por defeito.

Efectuando uma análise do modelo de objectivos para a produção do modelo de *features*, algumas variabilidades e decomposições das *features* podem já ser extraídas de modo a facilitar a identificação futura destas. Deste modo, utilizando a heurística C as *features* "Maq_Entrada" e "Maq_Saída" irão fazer parte de toda e qualquer configuração possível, visto que, havendo uma instância para o sistema, estas irão fazer parte do mesmo para permitir a entrada e saída dos veículos, logo estas serão obrigatórias no modelo. Verifica-se também que as *features* "Adesão_Sistema" e "Activação" serão opcionais, porque para uma configuração possível, não farão parte da variante do bilhete, fazendo parte apenas nas duas restantes variantes (identificador ou cartão). Uma vez que as *features* bilhete, identificador e cartão são usadas para aceder o parque nas suas respectivas máquinas, através da heurística A

estas irão fazer parte da decomposição destas máquinas a qual se encontram associadas, e estas *features* serão obrigatórias.

Deste modo, na Figura 5.14 é ilustrado o modelo de *features* com as *features* obtidas que se encontram na Tabela 5.10.

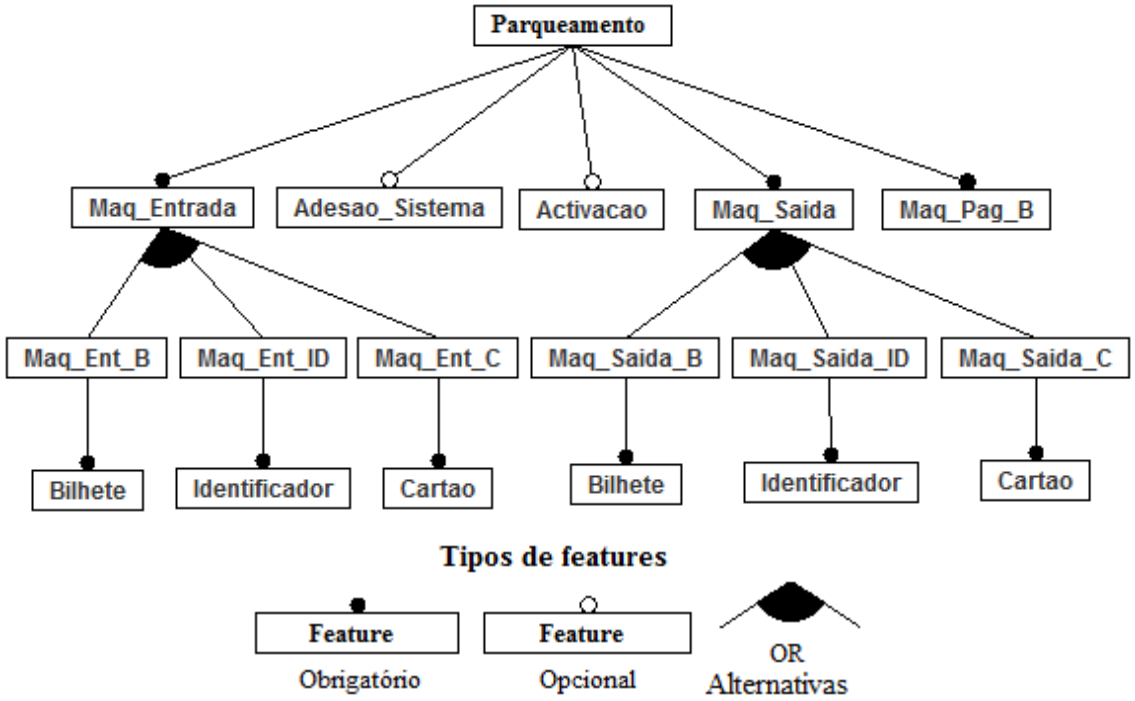


Figura 5.14 - Modelo de *features* incompleto -Versão 1 e legendas das notações.

Utilizando as heurísticas de obtenção das *features* do sistema, na Tabela 5.11 são descritas as *features* identificadas a partir dos requisitos do modelo de objectivos (Figuras 5.8, 5.9, 5.10, 5.11, 5.12 e 5.13) para o sistema. É de notar que algumas *features* identificadas, já foram obtidas a partir dos objectivos, reforçando a presença destas *features* no modelo de *features*.

Tabela 5.11 - Identificação de *features* a partir dos requisitos do Sistema.

Requisitos	*Features*	Descrição	H
-Registar a hora de entrada	-Maq_Entrada	-*Feature* "Maq_Entrada" já identificada.	1
-Acender a luz do semáforo	-Semáforo	-A comunicação com os utilizadores é bastante importante, neste caso um semáforo para interagir com os utilizadores.	1
-Abrir a cancela	-Cancela	-A cancela é mais uma propriedade a ter em conta, isto porque permite compor o sistema da portagem, limitando o acesso ao parque.	1
-Ler o cartão	-Cartão -Leitor (requerida)	- *Feature* "Cartão" já identificada. -É importante um leitor visto que necessário para a leitura dos cartões para aceder ao parque.	1,3
-Validar o cartão	-Cartão	-*Feature* "Cartão"já identificada.	1
-Detectar o	-Detecção do ID	-*Feature* "Identificador" já identificada.	1,2,3

identificador	-Identificador -Sensor (requerida)	-O "sensor" permite a detecção do identificador para aceder o parque.	
-Validar o identificador	-Identificador	-*Feature* "Identificador" já identificada.	1
-Calcular valor a pagar	--Maq_Pag_B	-*Feature* "Maq_Pag_B" já identificada.	1
-Mostrar o valor a pagar no visor	-Visor	-A comunicação com os utilizadores é bastante importante, neste caso um visor para interagir com os utilizadores é indispensável ao sistema.	1
-Registar hora de saída	-Maq_Saída	-*Feature* "Maq_Saída" já identificada.	1

Após a identificação das novas *features* através das heurísticas para a obtenção das *features* de um sistema definidas acima, estas são adicionadas ao modelo de *features* do sistema com a ajuda das heurísticas para o desenvolvimento do modelo de *features*, permitindo ter o modelo apresentado na Figura 5.15. Algumas *features* obtidas já se encontram representadas no modelo. Nesta figura é ilustrada a *feature* "Detecção do ID" que permite o início da actividade relacionada com o identificador. Esta vai fazer parte da decomposição das *features* "Maq_Ent_ID" e "Maq_Saída_ID", visto que ela despoleta a acção sobre a actividade relacionada com a entrada através do identificador. As variabilidades das outras *features* obtidas na tabela acima serão ilustradas por defeito como obrigatórias, de modo que as mesmas sejam especificadas na sub-actividade seguinte do processo.

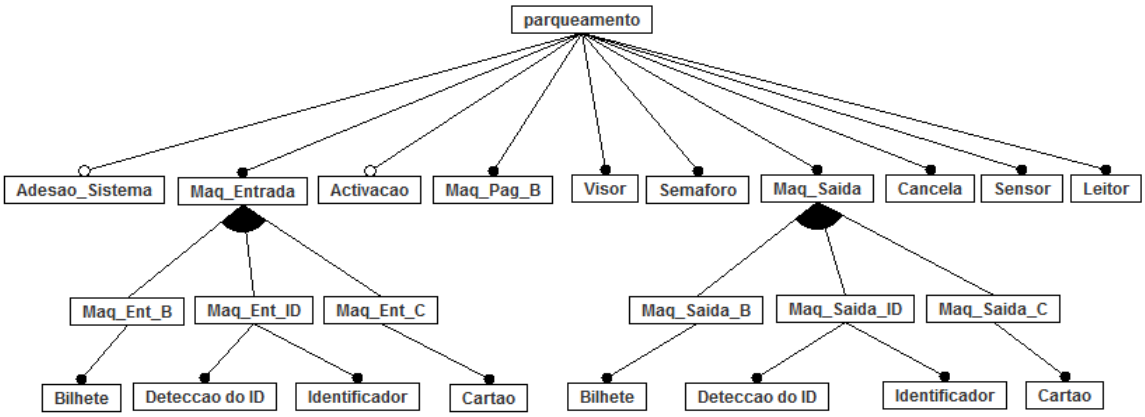


Figura 5.15 - Modelo de *features* incompleto – Versão 2.

As *features* identificadas a partir das expectativas do sistema (expectativas de todos os modelos de objectivos) através das heurísticas para o efeito são descritas na Tabela 5.12. Nesta tabela podemos encontrar também algumas *features* obtidas anteriormente, que já fazem parte do modelo de *features* em desenvolvimento.

Tabela 5.12 - Identificação de *features* a partir das Expectativas do Sistema.

Expectativas	*Features*	Descrição	H
-Fornecer dados pessoais	-Dados_Pessoais	-*Feature* referente aos dados pessoais do cliente para a adesão.	1
-Fornecer dados do cartão de débito	-Dados_CD	- *Feature* referente aos dados do cartão de débito do cliente para a adesão.	1
-Fornecer dados do veículo	-Dados_Veículo	- *Feature* referente aos dados do veículo do cliente para a adesão.	1
-Efectuar a activação do identificador	-Identificador -Activação (requerida)	-*Feature* "Identificador" já identificada e requer uma activação para a sua utilização.	1,3
-Efectuar a activação do cartão	-Cartão -Activação (requerida)	-*Feature* "Cartão" já identificada e requer uma activação para a sua utilização.	1,3
-Pressionar o botão da máquina de entrada	-Maq_Entrada	-*Feature* "Maq_Entrada" já identificada.	1
-Retirar bilhete da máquina de entrada	-Bilhete -Maq_Entrada (requerida)	-*Feature* "Cartão" já identificada. -*Feature* "Maq_Entrada" já identificada.	1,3
-Passar o cartão no leitor	-Cartão -Leitor (requerida)	-*Features* "Leitor" e "Cartão" já identificadas;	1,3
-Dirigir-se a cancela	-Cancela	-*Feature* "Cancela" já identificada.	1
-Inserir o bilhete na máquina de pagamento	-Maq_Pag_B -Bilhete (requerida)	-*Feature* "Bilhete" já identificada. -Para a saída do parque é necessário inserir o bilhete na máquina de pagamento para o efeito.	1,3
-Inserir a quantia a pagar	-Maq_Pag_B	-*Features* "Maq_Pagamento" já identificada;	1
-Retirar o bilhete na máquina de pagamento	-Bilhete -Maq_Pag_B (requerida)	-*Features* "Maq_Pagamento" e "Bilhete" já identificadas;	1,3
-Inserir o bilhete na máquina de saída	-Bilhete -Maq_Saída (requerida)	-*Features* "Maq_Saída" e "Bilhete" já identificadas.	1,3

A Figura 5.16 ilustra o modelo de *features* com as novas *features* obtidas para o sistema de parqueamento, através das heurísticas para o desenvolvimento do modelo de *features*. De lembrar que a maior parte das *features* identificadas na tabela acima já foram identificadas, o que permite reforçar o seu aparecimento no modelo de *features*. Deste modo, apenas as *features* (Dados_Pessoais, Dados_CD e Dados_Veículo) relacionadas com a adesão ao sistema não foram especificadas. Estas irão fazer parte da decomposição da *feature* "Adesão_Sistema", visto que os dados do cliente são necessários para adesão do sistema, tornando-os obrigatórios no modelo de *features* para o sistema.

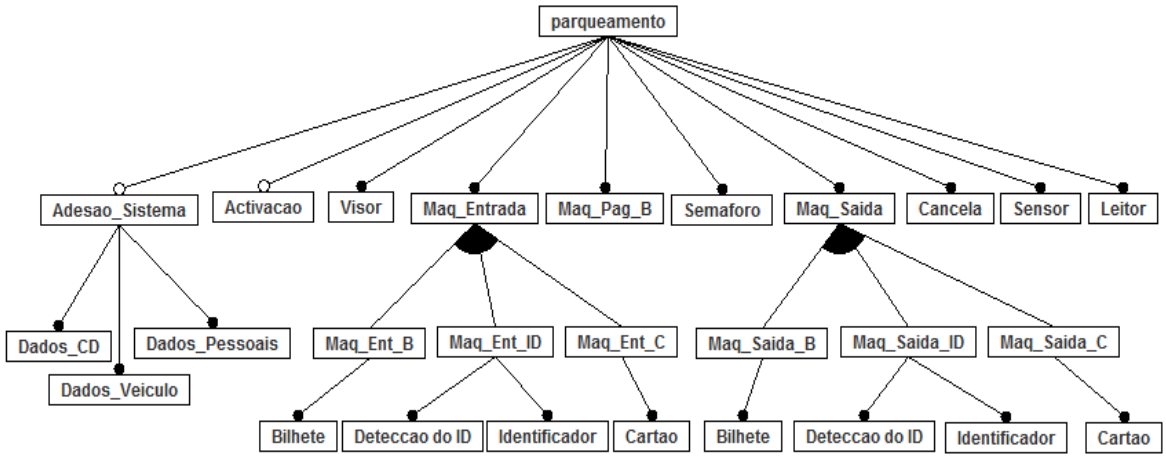


Figura 5.16 - Modelo de *features* incompleto - Versão 3.

De modo a estruturar melhor as *features* no modelo, eliminou-se a repetição destas tornando o modelo menos redundante. Estas repetições foram surgindo a medida que se foi identificando as *features* para o desenvolvimento do modelo. Deste modo a Figura 5.17 ilustra o modelo de *features* refinado para o sistema.

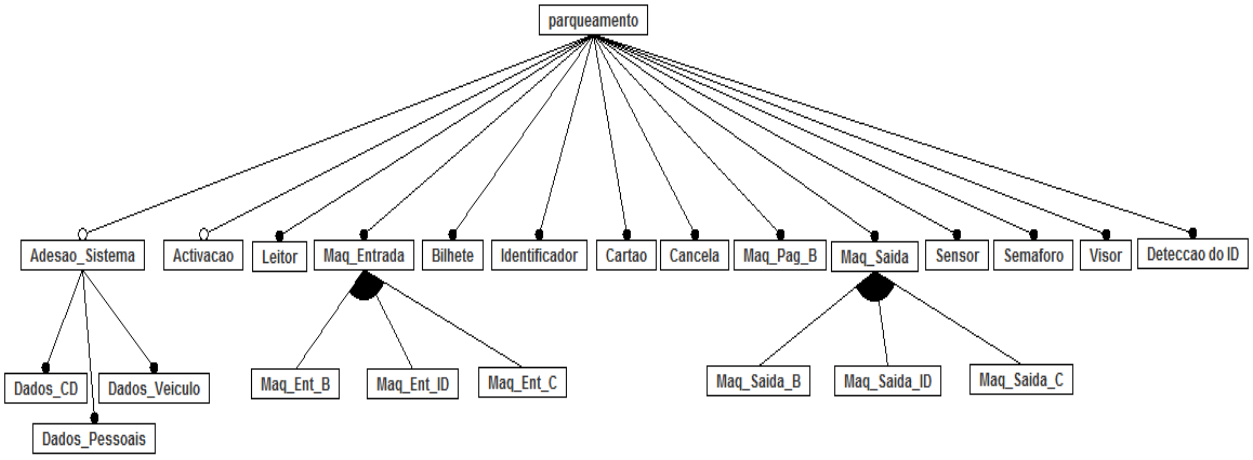


Figura 5.17 - Modelo de *features* incompleto – Versão 4.

Após a estruturação do modelo é necessário identificar o resto das variabilidades do sistema no modelo. Deste modo, na subsecção seguinte serão identificadas estas variabilidades e as interacções que podem existir entre as diferentes *features* existentes no modelo de *features*.

ii. Identificar as variabilidades do sistema e as interacções entre *features* do sistema

Quando produzimos um modelo de *features*, é importante realçar a variabilidade existente no mesmo, visto que esta é uma característica relevante para o sistema.

Na subsecção anterior foram definidas heurísticas que permite identificar as variabilidades de um sistema no modelo de objectivos para a LPS. Para a identificação, é essencial ter em conta as diferentes alternativas de se alcançar um objectivo do sistema descrito no modelo de objectivos, permitindo saber que *features* do sistema são obrigatórias ou opcionais. Outra propriedade a ter em conta é o documento que descreve o sistema, permitindo saber o que o cliente deseja.

Começando por identificar as variabilidades do modelo de *features*, é importante lembrar que determinadas *features* do sistema irão fazer parte de toda e qualquer configuração do sistema, estas são denominadas como obrigatórias. As opcionais podem ou não ocorrer em uma configuração possível.

Deste modo, a Figura 5.18 ilustra o modelo de *features* com as suas respectivas variabilidades. Nesta figura as *features* "Adesão_Sistema" e "Activação" encontram-se como opcionais porque só podem existir nas configurações que permitam a utilização do "Identificador" ou do "Cartão" através das máquinas de entrada e saídas dos mesmos. Analogamente acontece também para as *features* "Sensor" e "Leitor" que serão utilizadas apenas nas configurações usando o "Identificador" e "Cartão" pelas suas máquinas, respectivamente. Para o caso das *features* "Bilhete", "Identificador" e "Cartão" estas são opcionais visto que em certas configurações do sistema algumas não irão fazem parte das mesmas. O mesmo irá acontecer a feature "Detecção do ID" que apenas faz parte da configuração associada a máquina de entrada através do identificador. As *features* "Cancela", "Semáforo" e "Visor" são obrigatórias no sistema, visto que irão fazer parte de todas as instâncias do sistema.

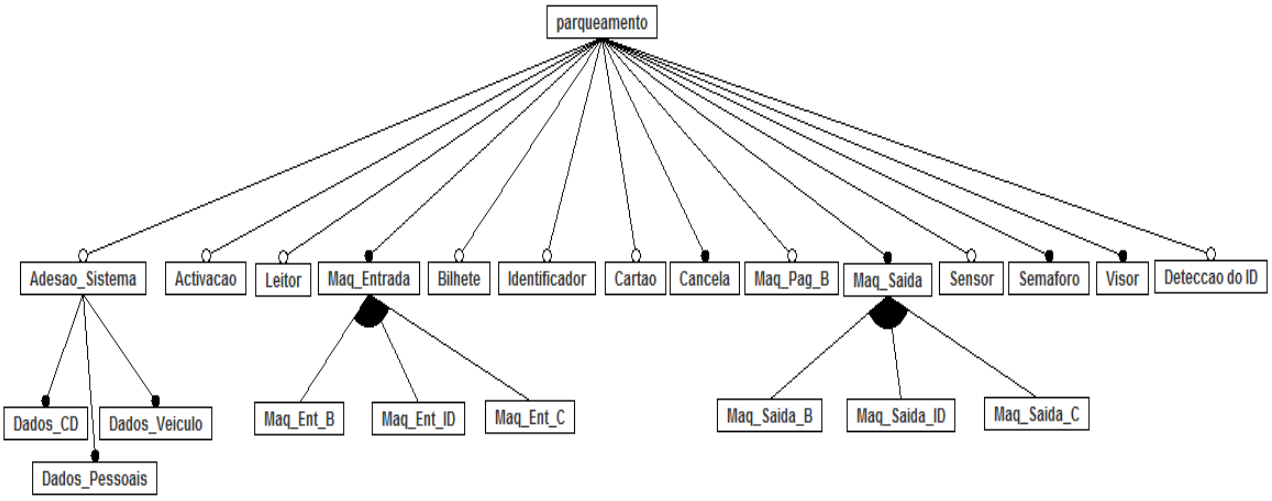


Figura 5.18 - Representação das variabilidades no modelo de *features*.

Uma vez especificadas as variabilidades do sistema no modelo de *features*, é essencial identificar as interacções existentes entre as *features* do sistema. Para a identificação destas interacções, é importante analisar os relacionamentos existentes entre os vários conceitos do modelo de objectivo, visto que as diferentes *features* foram obtidas a partir destes.

De acordo com a descrição do sistema na secção 5.2 e a especificação do modelo de objectivos, para a utilização do identificador e do cartão no sistema de parqueamento é necessária a adesão e posteriormente a activação dos mesmos. Deste modo, existe uma relação de dependência entre estas *features* no modelo, permitindo efectuar um relacionamento de *requires*.

As *features* "Adesão_Sistema" e "Activação" uma vez opcionais, estas relacionam-se entre si com ligações de *requires*, visto que para efectuar uma activação (identificador ou cartão) é necessário a adesão ao sistema. Para a utilização do identificador e do cartão é necessária a activação dos mesmos, permitindo haver uma relação de *requires* entre as features que as identificam.

É importante referir que as features "Maq_Ent_B e Maq_Saída_B", "Maq_Ent_ID e Maq_Saída_ID" e "Maq_Ent_C e Maq_Saída_C" relacionam com as features "Bilhetes", "Identificador" e "Cartão" através do relacionamento de *requires*, visto que para a utilização das máquinas é requerido o bilhete, o identificador ou o cartão consoante a forma de entrada no parque. De notar também que a *feature* "Detecção do ID" sendo opcional, apenas irá fazer parte da instância relacionada com o identificador. Assim esta *feature* é relaciona-se com as features "Maq_Ent_ID" e "Maq_Saída_ID" através da ligação de *requires*.

Para o caso das *features* "Sensor" e "Leitor", sendo estas opcionais, elas só pedem estar relacionadas com as *features* "Maq_Ent_ID e Maq_Saída_ID" e "Maq_Ent_C e Maq_Saída_C" respectivamente. Para estas interacções, o conceito de *requires* vai permitir relacionar estas *features* no modelo.

Deste modo, a Figura 5.19 descreve as interacções entre as *features* do sistema. As ligações de *requires* encontram-se representadas no modelo em forma de setas tracejadas.

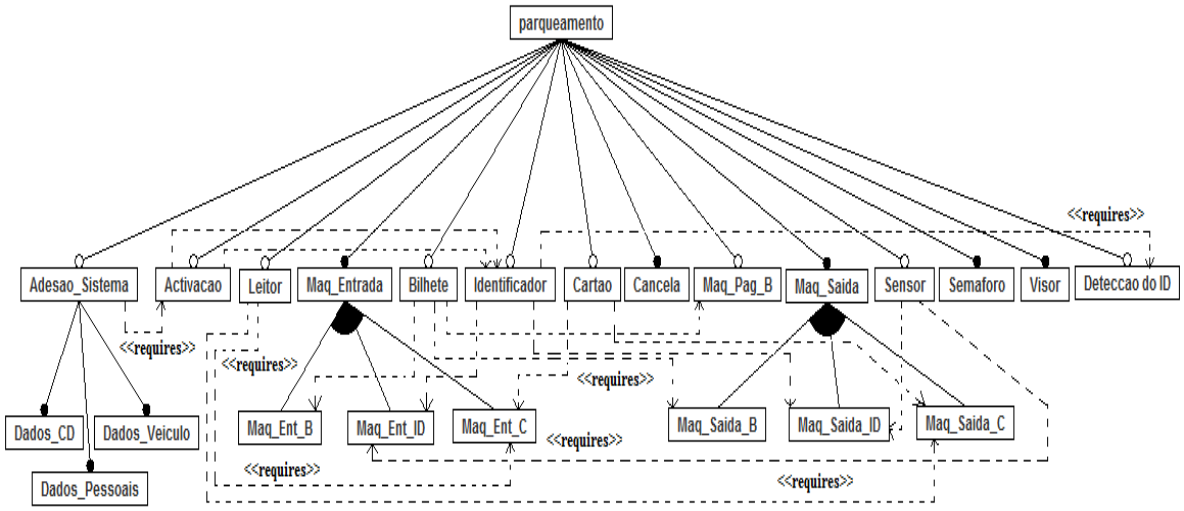


Figura 5.19 – Modelo de *features* completo e representação das interacções entre *features*.

Depois da modelação das *features* do sistema e da identificação das variabilidades e interacções do mesmo, é necessário efectuar a validações das decisões tomadas no desenvolvimento do LPS. Deste modo, na subsecção seguinte é descrita como é feita esta validação.

iii. Validar as *features* do sistema e as suas respectivas interacções

Para a validação das opções tomadas, isto é, validação das *features* do sistema e suas propriedades, é efectuada uma revisão minuciosa de modo a saber que o que se desenvolveu corresponde com as especificações desejadas. Estas revisões são efectuadas entre os clientes e os engenheiros de sistema. Esta actividade também não será tratada em detalhe neste trabalho.

Na subsecção seguinte é abordado o problema a nível da Engenharia da Aplicação.

5.2.2 A nível da Engenharia de Aplicação

Uma vez desenvolvido o artefacto base do sistema na Engenharia de Domínio, esta é utilizada na Engenharia de Aplicação para uma determinada instância do sistema. Nesta subsecção é configurada a LPS para uma aplicação.

5.2.2.1 Configuração do modelo de objectivos do sistema

De modo a apresentar uma configuração possível do modelo de objectivos, utilizou-se o caso do sistema de parqueamento da entrada e saída do parque através bilhete. Apenas irão ser apresentadas os modelos de objectivos que sofrem alterações. Como nesta aplicação a entrada

com o identificador e o cartão não são consideradas, os modelos referentes a estas situações serão excluídos. Na Figura 5.20 é apresentada a configuração do sistema para permitir a entrada no parque através do bilhete.

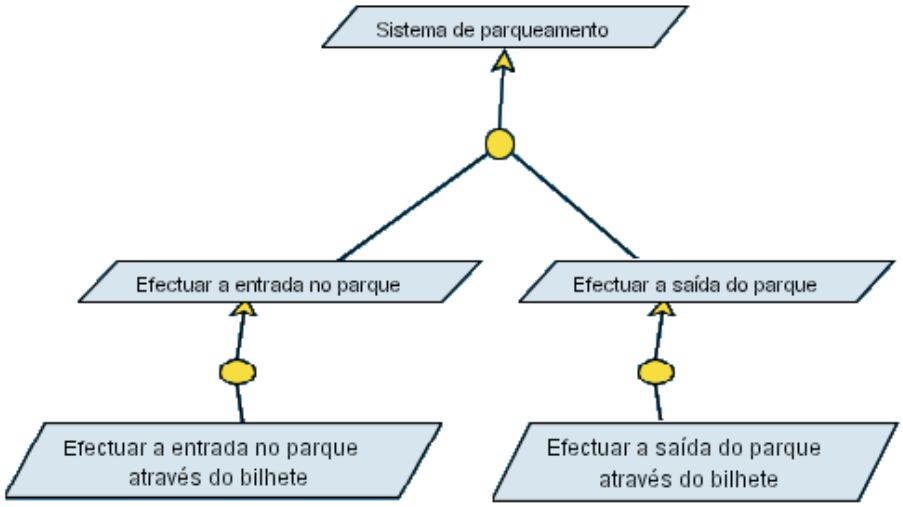


Figura 5.20 - Configuração do modelo de objectivos para aceder ao parque através do bilhete.

Seguidamente é efectuada a configuração da LPS para o modelo de *features* do sistema.

5.2.2.2 Configuração do modelo de *features* do sistema

Uma vez aplicada a configuração para o modelo de objectivos na Figura 5.21 é ilustrada a configuração para o modelo de *features* do sistema. Nesta configuração é descrita o caso para o acesso ao parque com o bilhete.

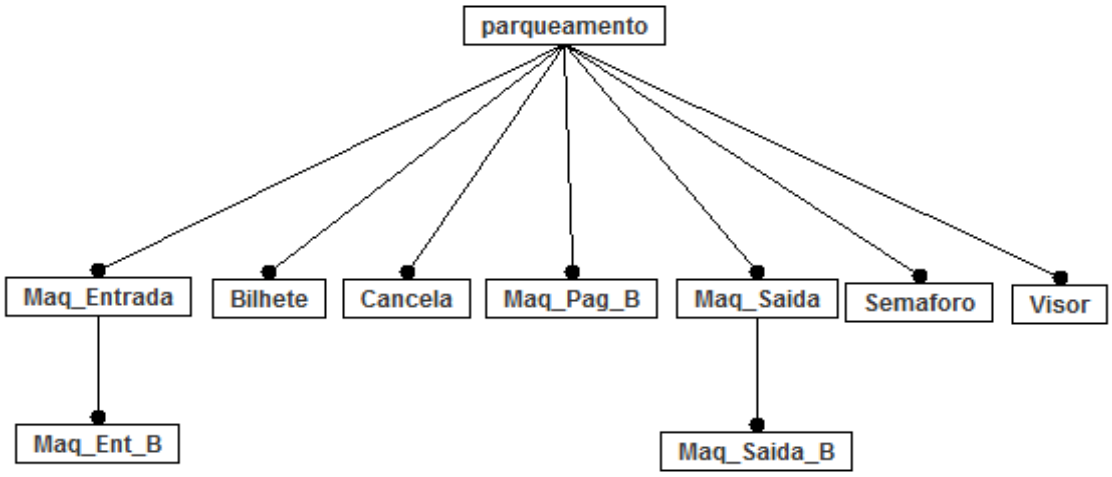


Figura 5.21 – Configuração para uma aplicação para aceder ao parque através do bilhete.

A configuração de um modelo de *features* também pode ser representada em forma de uma árvore de directórios [39]. Deste modo a Figura 5.22 ilustra a mesma para o exemplo em causa, onde se selecciona com uma marca as caixas antes das *features* que se deseja considerar em uma determinada configuração.

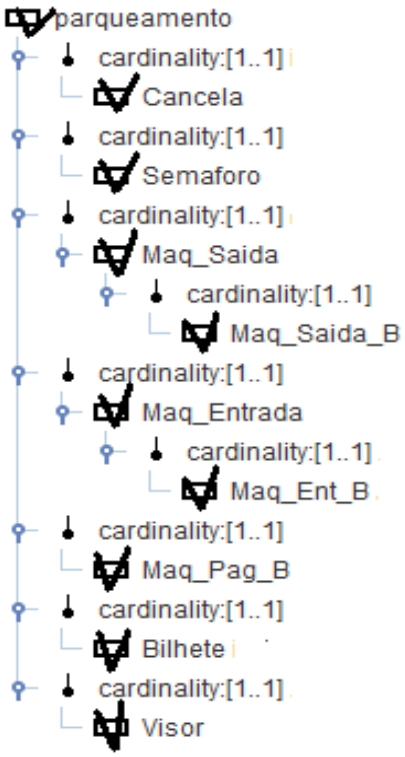


Figura 5.22 - Configuração da entrada com o bilhete em forma de árvore de directório.

Como se pode verificar nas duas configurações, nos modelos de *features* e de objectivos apenas fazem parte as features e os conceitos necessários para uma determinada instância da LPS.

Após a definição do processo para a abordagem LPS-KAOS, vamos estudar os metamodelos das duas abordagens em causa (abordagem KAOS e modelo de *features*) de modo a perceber melhor a interacção entre os elementos que fazem parte de cada modelo.

5.3 Integração do metamodelo para a abordagem LPS-KAOS.

De modo a ter uma melhor percepção do relacionamento entre a abordagem KAOS e o modelo de *features*, foi efectuada uma análise dos seus respectivos metamodelos. Um metamodelo descreve a estrutura de um modelo, isto é, como os elementos de um determinado modelo encontram-se relacionados entre si, permitindo um maior entendimento

das propriedades existentes no modelo [40]. Aqui vamos descrever os metamodelos através de diagramas de classe UML [7, 8].

Nesta secção vamos descrever os metamodelos dos modelos de features e KAOS que fazem parte da abordagem apresentada, permitindo consolidar o raciocínio aplicado na secção anterior. Em seguida, é efectuado um mapeamento entre os metamodelos para facilitar a transformação de modelos, relacionando as propriedades e conceitos existentes em cada metamodelo.

5.3.1 Metamodelo do modelo de *features*

Na Figura 5.23, é ilustrado o metamodelo proposto por Czarnecki *et al*. [33]. Segundo os autores, um modelo de *features* é composto por uma ou mais *features* raiz, que constituem a (s) raiz (es) dos diferentes diagramas de *feature*. A *feature* raiz é apenas um dos três tipos diferentes de *features*. As outras duas são a *feature* agrupada e a *feature* solitária (pode pertencer à agrupada). As *features* podem ter um atributo opcional de um certo tipo, e estes atributos podem ter um valor opcional (*integer* e *string*). Nesta figura, é ilustrada também uma classe denominada "*FDReference*" que representa uma referência do diagrama de *feature.* Esta classe pode referenciar apenas uma *feature* raiz, mas esta *feature* pode ser referenciada por diversas referências. Por exemplo, através do modelo de *features* do exemplo para ilustrar o processo para o desenvolvimento do LPS, podemos identificar a *feature* raiz "Parqueamento" que é referenciada por várias *sub-features* (por exemplo: Adesão_Sistema, Activação, Sensor, etc.). As classes abstractas "*ContainableByFG*" e "*ContainableByF*" representam classes que relacionam a(s) *feature*(s) que pode (m) ser contida (s) por um grupo de *feature* e uma *feature* respectivamente.

Um grupo de *feature* pode conter apenas *features* agrupadas ou referências de diagramas de *features*, por exemplo utilizando o exemplo do processo, pode-se constatar que no modelo de *features*, as *features* "Maq_Ent_B", "Maq_Ent_ID" e "Maq_Ent_C" encontram--se agrupadas com ligação do tipo "OR", este grupo de *features* pertencem a *feature* "Maq_Entrada" que é composta apenas por elas. Uma *feature* pode conter somente *features* solitárias, grupos de *features,* e referências. Por exemplo, para o caso da *feature* "Adesão_Sistema", esta é composta por um grupo de *features* (Dados_Pessoais, Dados_CD e Dados_Veículo) e é referenciada pela *feature* "Activação". Uma *sub-feature* é qualificada por uma cardinalidade de *feature* que é fixa ao relacionamento da *sub-feature* solitária e que representa o número de vezes que esta pode surgir na configuração do sistema.

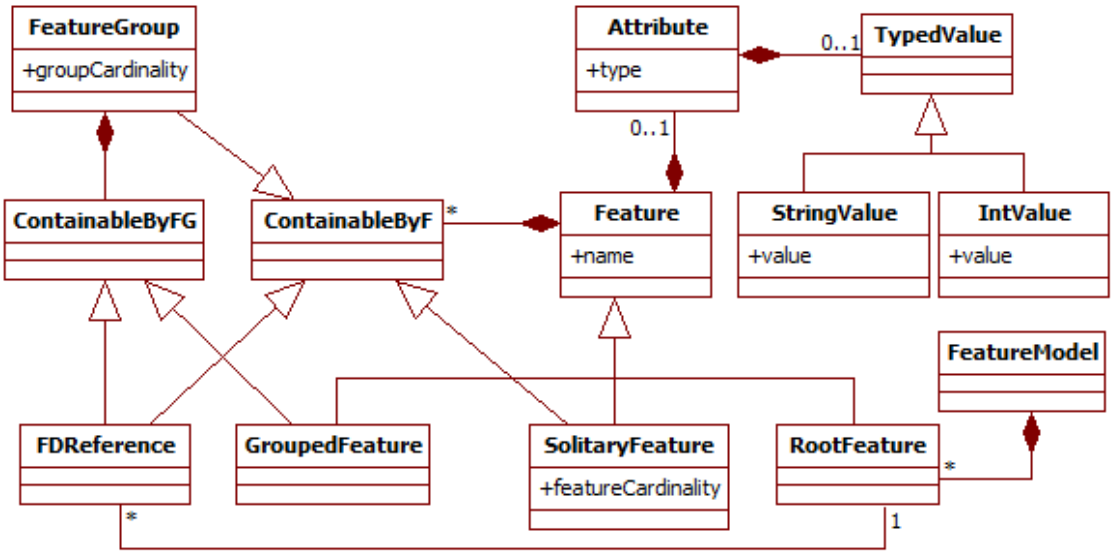


Figura 5.23 - Metamodelo do Modelo de *features* [33].

Seguidamente é efectuada uma descrição do metamodelo do modelo de objectivo da abordagem KAOS.

5.3.2 Metamodelo do modelo de objectivos da abordagem KAOS

Como foi descrito no capítulo 2, a abordagem KAOS é composta por quatro modelos importantes que por sua vez possuem várias propriedades que se encontram descritas no metamodelo. Uma vez que neste capítulo é focado com maior pormenor o modelo de objectivos, apenas será efectuado um estudo parcial do metamodelo KAOS que evidencia os seus conceitos e propriedades do modelo de objectivos. Deste modo, na Figura 5.24 é ilustrado este metamodelo.

Nesta figura os objectivos (*goals*) podem ser classificado de acordo com um dos 4 padrões: conservar (*Maintain*), contornar (*Avoid*), atingir (*Achieve*) e cessar (*Cease*). Os objectivos de conservação requerem que algumas propriedades estejam asseguradas. Os objectivos a contornar requerem que uma determinada propriedade nunca estejam asseguradas. Os atingidos requerem que alguma propriedade esteja eventualmente assegurada e os de cessar requerem eventuais paragens para assegurar [40].

O objectivo pode ser refinados através de ligações de refinamento (*G-Refinament*), onde relaciona-o com um conjunto de sub-objectivos. Estas ligações são relacionamentos do tipo AND/OR e apresentam um conjunto de alternativas de sub-objectivos que contribui para a

satisfação do objectivo pai. Um objectivo pode ter *G-refinaments* onde resulta em diferentes desenhos de *software* [40].

O *Softgoal* está relacionado com as características não-funcionais de um sistema. É um objectivo que não tem um critério claro para a sua satisfação. A satisfação de objectivo é introduzida para expressar que os sub-objectivos são esperados para contribuir para a satisfação do *softgoal* dentro de limites aceitáveis. Os *softgoals* podem ser refinados como qualquer outro objectivo do KAOS e conflitos entre eles podem ser capturados [40].

Um agente (*Agent*) é um objecto activo onde executa um determinado papel para alcançar um objectivo por controlo específico do ambiente do objecto. O relacionamento de atribuição (*Assignment*) é definido como designação possível de um objectivo para um agente. A ligação de responsabilidade (*Responsability*) define uma atribuição actual de um objectivo a um agente. Um objectivo efectivamente atribuído a uma agente de software (*Software Agent*) é denominado por um requisito (*Requirement*). Um objectivo atribuído a um agente do meio ambiente (*Environment Agent*) é denominado por uma expectativa (*Expectation*). Um agente monitoriza e controla um objecto se o estado do objecto é directamente observável ou controlado pelo agente [40].

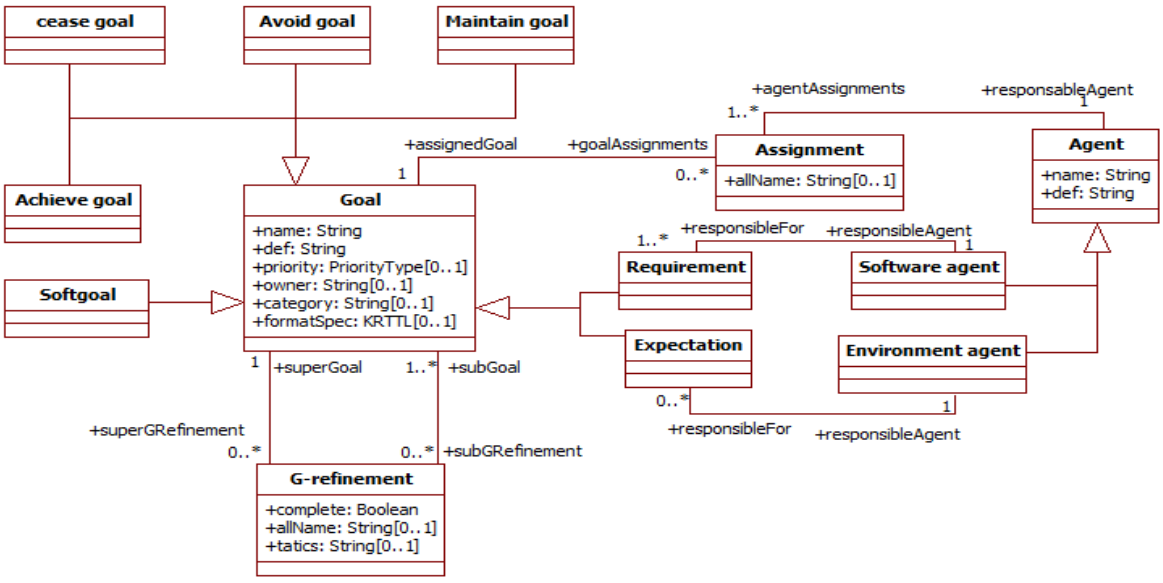


Figura 5.24 - Metamodelo do modelo de objectivos [40].

Na secção seguinte é efectuado um mapeamento entre os metamodelos do modelo de *features* e do modelo de objectivos da abordagem KAOS.

5.3.3 Relacionamento entre os metamodelos do modelo de *features* e modelo KAOS

As *features* são propriedades do sistema usadas para distinguir os requisitos comuns dos variáveis de um determinado sistema, derivadas dos objectivos, requisitos ou expectativas do modelo de objectivos deste sistema. Assim, existe uma ligação entre as propriedades dos dois metamodelos que facilita o relacionamento.

Uma vez que foi efectuada uma análise dos dois metamodelos separadamente, a Figura 5.25 apresenta o metamodelo da abordagem LPS-KAOS que relaciona vários elementos do metamodelo do modelo de *features* com o metamodelo do modelo KAOS. As *features* podem estar associadas a um conjunto de requisitos de sistema, e também são propriedades de um sistema usadas para distinguir os requisitos comuns, dos requisitos variáveis de uma LPS; os objectivos podem expressar funcionalidades do sistema, que são decompostas até serem identificados os requisitos que fazem parte de um sistema. Deste modo, as *features* podem estar relacionadas a um ou mais objectivos, e um objectivo só pode estar associado a uma ou mais *features*. Por exemplo, a *feature* "Identificador" relaciona-se com vários objectivos, tais como, "Detectar identificador", "Validar Identificador", e no caso do objectivo "Ler o cartão" é possível extrair a *feature* "Cartão" e a *feature* requerida "Leitor" que permite a leitura do cartão.

Para além dos relacionamentos presentes em cada metamodelo (metamodelos do modelo de objectivos e modelo de *features*), existem relações entre classes destes, entre elas a classe *Feature* do modelo de feature com as classes *Goal* (objectivo), *Requirement* (requisito) e *Expectation* (expectativa) do modelo de objectivos. Estas ligações permitem demonstrar a interacção existente entre propriedades de cada metamodelo. Estas interacções ilustram os relacionamentos seguintes:

- Uma *Feature* está associada com um ou mais objectivos;
- Um objectivo está associado com zero ou mais *Features*;
- Uma *Feature* está associada com zero ou mais requisitos;
- Um requisito está associado com zero ou mais *Features*;
- Uma *Feature* está associada com zero ou mais expectativas;

- Uma expectativa está associada com zero ou mais *Features*.

Deste modo, a Figura 5.25 ilustra o relacionamento existente entre as classes dos dois metamodelos, permitindo definir o metamodelo para a abordagem proposta.

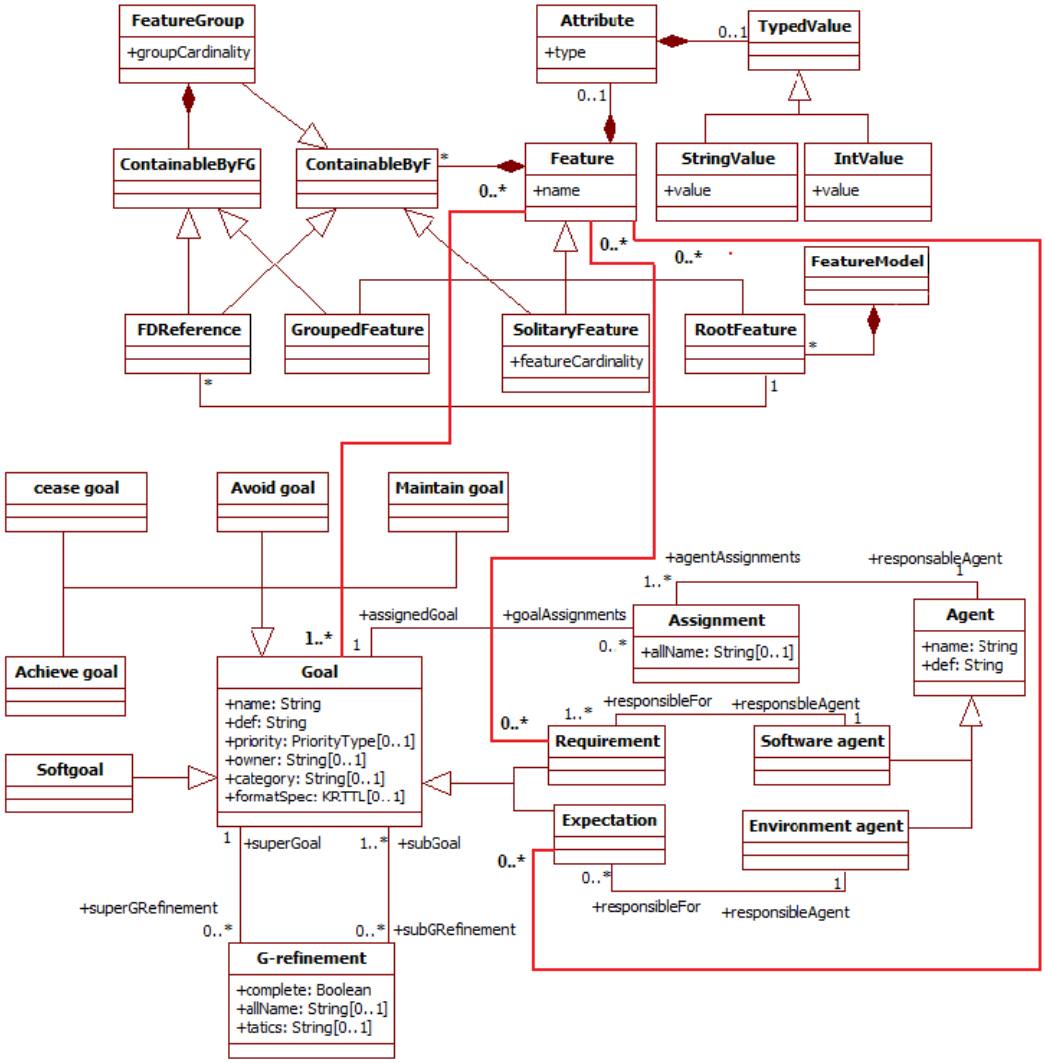


Figura 5.25 – Metamodelo da abordagem LPS-KAOS.

De modo a ter uma melhor visão e percepção dos relacionamentos entre os dois metamodelos, será utilizado exemplos de sistemas para estes relacionamentos, isto é, serão utilizados exemplos de casos que podem não terem sido contemplados nos sistemas desenvolvidos nos capítulos desta dissertação. Para cada um dos relacionamentos citados acima, será ilustrado um exemplo prático:

- Uma *Feature* está associada com um ou mais objectivos

Ao aplicar o exemplo da "Via Verde" o objectivo de negócio "Sistema de Via Verde" da abordagem KAOS está relacionado a uma *feature* conceptual (*RootFeature*) que tem como nome "Parque" no modelo de *features*, visto que estas duas propriedades representam o sistema a ser desenvolvido em cada abordagem, isto é, no topo dos diagramas de cada modelo. Outro caso elucidativo do sistema de parqueamento, é o objectivo "Efectuar a adesão" no modelo de objectivos, que através deste obtém-se uma *feature* denominada "Adesão", isto é, a partir de um objectivo do sistema obtém-se uma *feature* para o sistema a desenvolver.

Para o caso de uma *feature* estar relacionada com mais do que um objectivo, os objectivos "Efectuar a adesão ao identificador" e "Efectuar a activação do identificador" permitem identificar a uma *feature* denominada "Identificador". De um modo geral, vários objectivos podem originar ou estar associados a uma determinada *feature*, visto que estes são os objectivos de um sistema que são descritos na fase inicial do desenvolvimento do mesmo.

- Um objectivo está associado com zero ou mais *Features*

Para este tipo de relacionamento, um objectivo pode não originar nenhuma *feature*, uma vez que num dado modelo pode não existir objectivos que originem *features*. Uma vez que os exemplos ilustrados foram apenas para dar uma visão mais concisa das propriedades, é possível que no sistema exista um objectivo que permita efectuar o cálculo da quantia, consentindo a não extracção de uma feature a partir do mesmo.

Neste relacionamento também podemos ter um objectivo associado a várias *features* de um sistema, como por exemplo o objectivo "Efectuar pagamento do bilhete", tanto está associado a uma *feature* denominada "Pagamento" como a uma *feature* "Bilhete" que permite os clientes a efectuarem o pagamento do bilhete para efectuar a saída do parque.

- Uma *Feature* está associada com zero ou mais requisitos

Para este caso, uma *feature* pode estar associado a nenhum requisito. Isto verifica-se quando temos *features* que são extraídas apenas de objectivos como foi descrito no relacionamento de uma *feature* com um ou mais objectivos. Como por exemplo, num sistema em que exista as *features* "P. Acumulado" e "P. Imediato" são *features* que não foram extraídas para dar uma descrição mais notável da distinção dos tipos de pagamentos no sistema de parqueamento. De notar que para as features e objectivos, o mesmo pode acontecer.

Outra propriedade a ter em conta neste relacionamento, é o facto de uma *feature* referir a um ou mais requisitos de um sistema. Por exemplo, a *feature* "Sensor" é obtida através do requisito "Detectar Identificador" do sistema, que permite a detenção de um identificar. A *feature* "Identificador" está relacionada com os objectivos "Detectar Identificador" e "Validar Identificador" do sistema.

- Um requisito está associado com zero ou mais *Features*

Esta relação permite que um requisito possa ou não estar associado a uma *feature*. Por exemplo os requisitos "Calcular valor a pagar" e "Registar a hora de saída" na saída do parque de estacionamento não permite obtenção de nenhuma *feature* para o sistema.

Com o requisito "Detectar Identificador" é extraído a *feature* "Sensor" e também o "Identificador", permitindo que a obtenção de duas *features* a partir de um requisito.

- Uma *Feature* está associada com zero ou mais expectativas

Com este relacionamento é possível associar as *features* às expectativas. Na obtenção de uma *feature* pode não estar presente nenhuma expectativa, contendo apenas objectivos ou requisitos como foi descrito no relacionamento de uma *feature* estar associada a zero ou mais requisitos.

Uma *feature* também pode ser obtida através de várias expectativas, como por exemplo, a partir da expectativa "Dirigir-se a cancela" obtém-se a *feature* "Cancela" para o sistema. Outro exemplo, com as expectativas "Pressionar o botão da máquina de entrada" e "Retirar o bilhete da máquina de entrada" pode-se obter uma *feature* denominada "Maq_Entrada".

- Uma expectativa está associada com zero ou mais *Features*

Para este relacionamento, podem existir requisitos que não dão origem a uma feature. Como por exemplo num outro sistema, a expectativa "Dirigir-se a zona de abastecimento" para o abastecimento de um veículo num sistema de abastecimento em que o veículo tem de se dirigir ao local de abastecimento de combustível, não interessa para a obtenção de uma *feature* para o sistema. Deste modo, é possível ter uma expectativa num determinado sistema que não dá origem a nenhuma *feature*.

No sistema do caso de estudo do processo, a expectativa "Passar o cartão no leitor" origina as features "Cartão" e "Leitor" para o sistema. Assim uma expectativa pode estar associada com várias *features* do sistema.

De um modo geral, numa 1ª fase para a obtenção das *features* de um sistema são necessárias a extracção dos objectivos de um sistema. Na abordagem KAOS os objectivos dos sistemas são obtidos em forma de requisitos, facilitando posteriormente a aquisição das *features* desejadas para sistemas. Deste modo, conhecer primeiro os objectivos do sistema, permite aos *stakeholders* um melhor entendimento do que se pretende do sistema e prontificar uma aquisição mais precisa das *features* do sistema.

De notar que para esta dissertação não foi possível contemplar toda a abordagem KAOS para a LPS, ficando por abranger outras propriedades da abordagem, entre elas uma possível relação de *features* e *softgoals*, os outros modelos pertencentes a mesma, conflitos, obstáculos, etc. Para estes casos não abordados, sugere-se para investigações futuras de modo a complementar o estudo sobre a abordagem KAOS para as LPS. De qualquer forma os modelos de objectivos satisfazem de forma significativa o que é proposto neste trabalho, facilitando a definição do modelo de *features*.

5.4 Resumo

Neste capítulo foi descrita a abordagem LPS-KAOS para a especificação das Linhas de Produto de *Software*. A abordagem é composta por dois processos: a nível da Engenharia de Domínio e a nível da Engenharia da Aplicação.

No processo de Engenharia do Domínio são definidos um conjunto de actividades para o tratamento da abordagem. Estas actividades consistem na Identificação e identificação dos objectivos do sistema, em que são obtidos os objectivos de um sistema a desenvolver, e na identificação e especificação da *features* do sistema que permite a obtenção da *features* de um sistema para uma determinada Linha de Produto de *Software*.

A nível da Engenharia da Aplicação é efectuada uma configuração do modelo de features e do modelo de objectivos da abordagem KAOS da LPS em causa para uma determinada instância da aplicação.

Para além destes processos, foi efectuada uma descrição dos metamodelos das duas abordagens de modo a facilitar a integração da abordagem KAOS para o desenvolvimento de

Linhas de Produtos de *Software*, relacionando as diferentes propriedades e conceitos existentes nos metamodelos.

6 Caso de estudo e comparação com outras abordagens

Neste capítulo é apresentado um caso de estudo real denominado sistema de Saúde Pública, onde será aplicado a abordagem LPS-KAOS. Seguidamente, uma comparação entre a abordagem LPS-KAOS e outras abordagens será efectuada.

6.1 Caso de estudo "Health - Watcher"

Neste capítulo será elaborado um caso de estudo real para ilustrar a abordagem [41]. Pretende-se o desenvolvimento de um sistema de saúde pública denominado observador de saúde (*Health – Watcher*) que permite assegurar a qualidade de serviços das instituições de cuidado de saúde. Este caso de estudo baseia-se num sistema real da indústria. Os requisitos do sistema são descritos da seguinte maneira [41]:

"O sistema proposto é para coleccionar e controlar as reclamações e notificações, fornecendo também importante informação para as pessoas, sobre o sistema de saúde. Com o desenvolvimento deste sistema, o sistema de saúde pública fornecerá consideravelmente:

- O controlo das reclamações;
- Qualidade de serviço na divulgação da informação.

O cidadão poderá ter acesso ao sistema pedido informação sobre os serviços de sistema ou efectuar a sua reclamação. Exemplo de perguntas que um cidadão pode fazer: informação sobre doenças (descrição, sintomas, prevenção de doenças), campanhas de vacinação, informação sobre reclamações feitas pelo cidadão (tais como, nome do reclamante, data da reclamação, descrição e observações da reclamação, situação (aberta, suspensa, fechada), análise técnica, data da análise, empregado que fez análise), onde as unidades de saúde tratam

de uma determinada especialidade, de onde são estas especialidades em uma unidade de saúde.

No evento de uma reclamação, esta estará registada no sistema e endereçada para um departamento específico (representado por um assistente registado), onde será permitido cuidar o procedimento e devolver uma resposta quando a análise estiver completa. Esta solução será registada no sistema, estando disponível para outras perguntas. Os tipos de reclamações são:

- Reclamação de Animais: apreensão de animais, controlo de vectores (roedores, escorpiões, morcegos, etc.), doenças relacionadas com mosquitos, animais maltratados. Precisa-se de informação adicional: tipo de animal, quantidade de animais, data do distúrbio, localização do distúrbio;
- Reclamação alimentar: casos onde é suspeito a ingestão de alimentos infectado. Informação adicional: nome da vítima data da vítima; quantidade de pessoas que comeram a refeição; quantidade de pessoas doentes; quantidade de pessoas que serão enviadas ao hospital e quantidade de pessoas falecidas; localização onde os pacientes estão em tratamento; refeição suspeita;
- Reclamação diversa: casos relacionados à varias razões, onde não foram mencionadas acima (restaurantes com problemas de higiene, vazamento da rede de esgoto, aguas suspeitas transportadas em camiões, etc.).

Os produtos serão postos também para o uso público em quiosques, em vários pontos estratégicos onde os próprios cidadãos efectuarão reclamações e receber informações. Também, o novo sistema deve permitir troca de informação com o Sistema de Vigilância Sanitária (SVS). Finalmente, um relatório com estatísticas que mostra a frequência de tipos de reclamações, e é enviada para o director de unidade de saúde, todos os meses."

Uma vez tida a descrição dos requisitos do sistema, inicia-se o desenvolvimento da LPS do sistema através do processo definido no capítulo 4. Na secção seguinte é descrito o sistema a nível da Engenharia do Domínio.

6.1.1 Engenharia de Domínio

Através das actividades definidas na abordagem LPS-KAOS para especificar o LPS, é elaborado a LPS a nível da Engenharia do Domínio.

6.1.1.1 Identificação e especificação dos objectivos do sistema

Para este caso de estudo foram identificados os seguintes objectivos principais do sistema:

- Efectuar pedido de informação ao sistema
- Efectuar registo do assistente no sistema
- Efectuar registo da reclamação no sistema
- Efectuar envio das estatísticas ao director da unidade de saúde
- Efectuar troca de informação com o SVS.

É importante referir que alguns destes objectivos podem ser decompostos em vários sub-objectivos alternativos, permitindo representar as várias opções de se alcançarem estes objectivos através do refinamento do tipo OR no modelo de objectivo. Na Tabela 6.1, é descrito como estes objectivos encontram-se decompostos em vários sub-objectivos.

Tabela 6.1 - Alguns objectivos e sub-objectivos (alternativas) do sistema.

Objectivos	Sub-Objectivos (alternativas)	Descrição
- Efectuar pedido de informação ao sistema	1) Efectuar pedido de informação sobre doenças; 2) Efectuar pedido de informação sobre campanha de vacinação; 3) Efectuar pedido de informação sobre unidades de saúde/especialidades; 4) Efectuar pedido de informação sobre reclamações.	- O cidadão pode optar pelo pedido de diversas de informação, tais como, sobre doenças, campanhas de vacinação, sobre unidades de saúdes e sobre reclamações existentes.
- Efectuar registo da reclamação no sistema	1) Efectuar registo da reclamação animal no sistema; 2) Efectuar registo da reclamação alimentar no sistema; 3) Efectuar registo da reclamação diversa no sistema.	- É possível efectuar registo de reclamação de três tipos diferentes: animal, alimentar e diversas.

Por sua vez, o sub-objectivo "Efectuar pedido de informação sobre unidades de saúde/especialidades" encontra-se decomposto em "Efectuar listagem das unidades de saúde por especialidades" e "Efectuar listagem das especialidades por unidades de saúde".

Deste modo, na Figura 6.1 é apresentado o modelo de objectivos do sistema de saúde pública com os objectivos e sub-objectivos identificados.

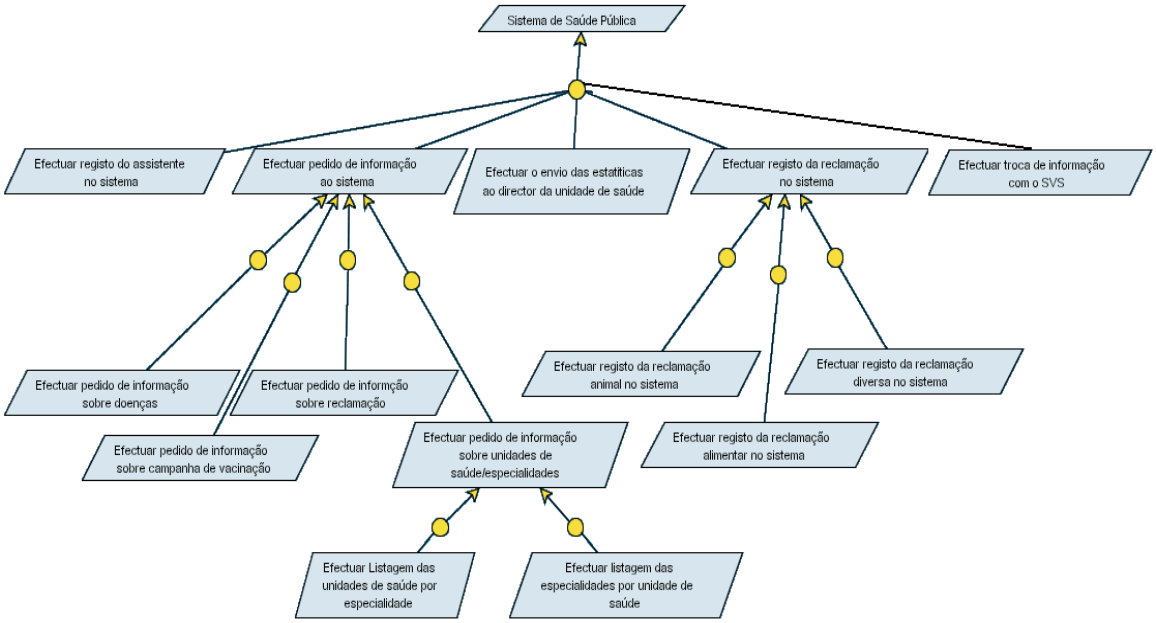


Figura 6.1 - Sistema de saúde pública (Objectivos e Sub-Objectivos).

Depois da obtenção dos objectivos e sub-objectivos do sistema, é possível identificar alguns requisitos e expectativas do sistema para que estes sejam alcançados. Portanto, para o caso do objectivo "Efectuar registo do assistente no sistema" tem-se como expectativa "Fornecer dados do assistente" que permite a introdução dos dados do assistente para o seu registo no sistema, e o requisito "Guardar dados do assistente" para a conclusão deste mesmo registo.

Uma vez que os dados são fornecidos pelo assistente e estes são armazenados no sistema pelo sistema de controlo, é possível identificar alguns agentes do sistema, tais como o "Assistente" e "Sistema de controlo" que permitem estar sob a responsabilidade da expectativa "Fornecer dados do assistente" e do requisito "Guardar dados do assistente", respectivamente.

Deste modo, a Figura 6.2 ilustra o modelo de objectivo que descreve como este objectivo pode ser alcançado no sistema.

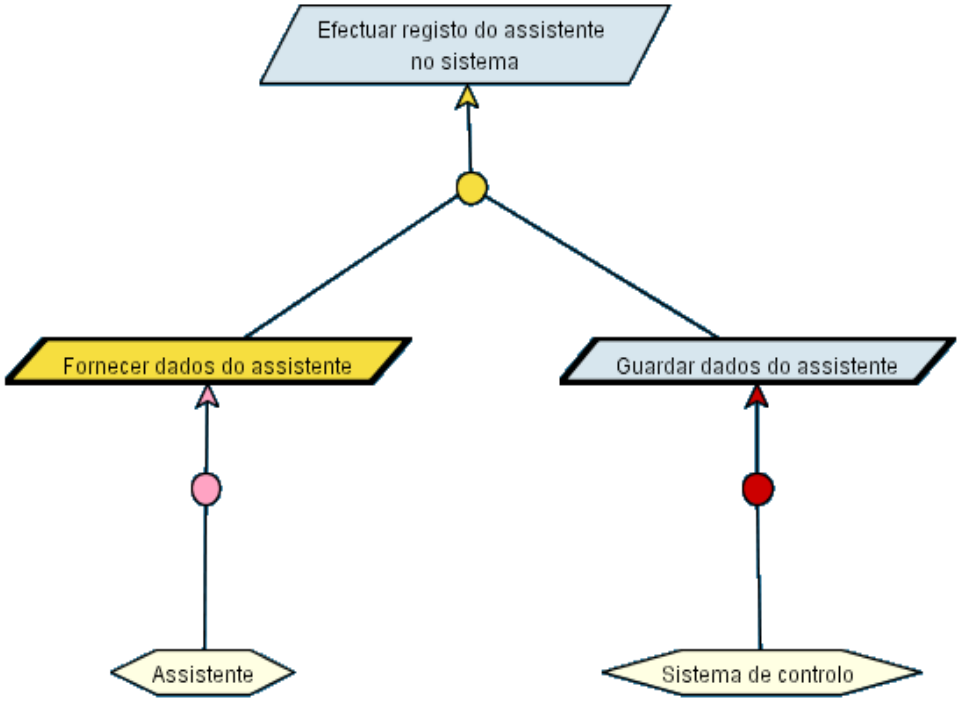


Figura 6.2 - Objectivo "Efectuar registo do assistente no sistema" e sua decomposição.

Como é descrita na Tabela 6.1, o objectivo "Efectuar pedido de informação ao sistema" encontra-se decomposto em quatro sub-objectivos. Começando por analisar os mesmos, estes também são refinados em alguns requisitos e expectativas do sistema. Deste modo, a Tabela 6.2 descreve estes requisitos e expectativas para o sub-objectivo "Efectuar pedido de informação sobre doenças".

Tabela 6.2 - Requisitos e Expectativas do objectivo "Efectuar pedido de informação sobre doenças".

	Efectuar pedido de informação sobre doenças	**Descrição**
Expectativas	-Seleccionar opção "Info-doenças" na página; -Seleccionar doença desejada.	-Para o pedido de informação sobre doenças, o cidadão selecciona o local referente as doenças na página. Posteriormente, após ser mostrada a lista de doenças disponíveis, selecciona a doença desejada.
Requisitos	-Mostrar lista de doenças disponíveis; -Mostrar informação sobre a doença seleccionada.	-Uma vez seleccionado o local referente as doenças no site, é apresentada uma lista de doenças disponíveis. Após ser seleccionada a doença desejada é exibida toda a informação referente a mesma.

Na Figura 6.3 é demonstrada a representação gráfica da decomposição do objectivo "Efectuar pedido de informação sobre doenças". As expectativas identificadas neste objectivo estão sob a responsabilidade de um agente denominado "Cidadão", visto que o cidadão selecciona opções desejadas para a obtenção da informação no sistema. Para os requisitos obtidos estes

encontram-se sob a responsabilidade do "Sistema de controlo" que despoleta a acção para exibir a informação desejada pelo cidadão.

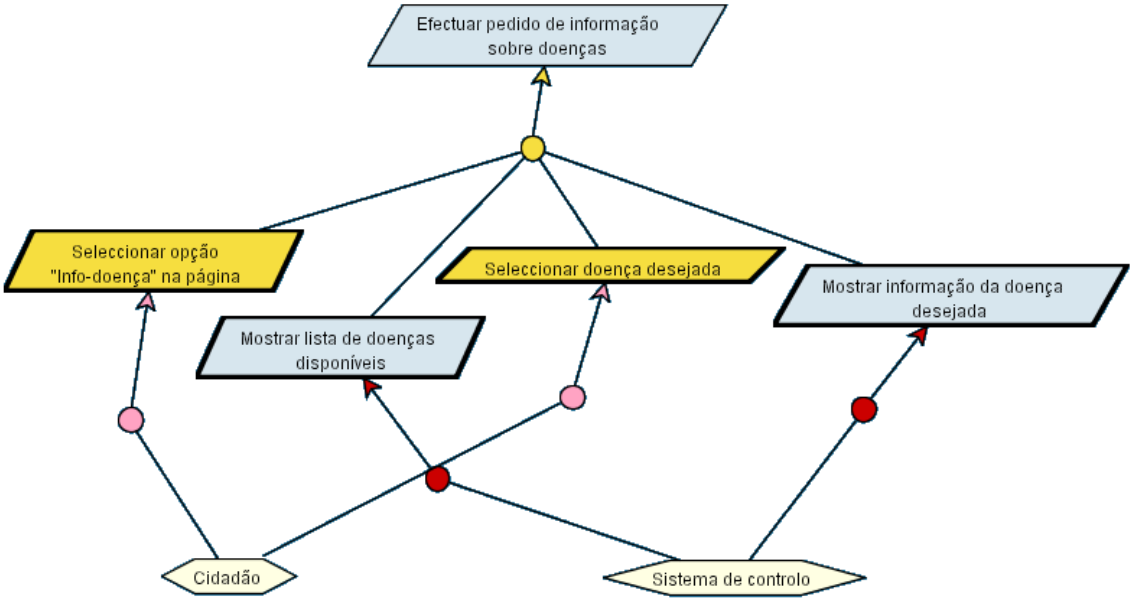


Figura 6.3 - Objectivo "Efectuar pedido de informação sobre doenças" e sua decomposição.

Na Tabela 6.3 são descritos os requisitos e as expectativas do objectivo "Efectuar pedido de informação sobre campanhas de vacinação".

Tabela 6.3 - Requisitos e Expectativas do objectivo "Efectuar pedido de informação sobre campanha de vacinação".

	Efectuar pedido de informação sobre campanha de vacinação	**Descrição**
Expectativas	-Seleccionar opção "info-campanha de vacinação" na página; -Seleccionar campanha de vacinação desejada.	-Para o pedido de informação sobre as campanhas de vacinação, o cidadão selecciona o local referente as campanhas na página. Posteriormente, após ser mostrada a lista de campanhas disponíveis, selecciona a desejada.
Requisitos	-Mostrar lista de campanhas de vacinação; -Mostrar informação sobre campanha de vacinação seleccionada.	-Uma vez seleccionado a opção referente as campanhas no site, é apresentada uma lista das mesmas. Após ser seleccionada a campanha desejada é exibida toda a informação referente a mesma.

Seguidamente, a Figura 6.4 ilustra o modelo de objectivos referente ao objectivo "Efectuar pedido de informação sobre campanhas de vacinação". De notar que para este caso, é também referenciado os dois agentes (Cidadão e Sistema de controlo) citados no objectivo "Efectuar pedido de informação sobre doenças" que se encontram sob a responsabilidade dos requisitos e das expectativas em causa.

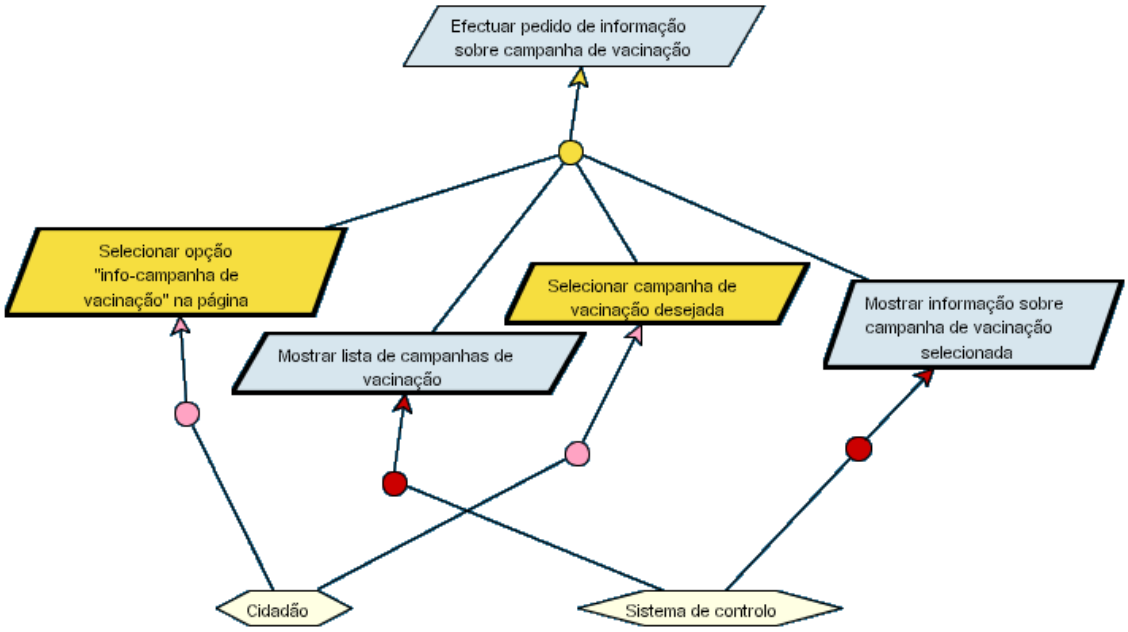


Figura 6.4. - Requisitos e Expectativas do objectivo "Efectuar pedido de informação sobre campanha de vacinação".

Continuando o mesmo raciocínio dos sub-objectivos identificados das Tabelas Tabela 6.2 e Tabela 6.3, na Tabela 6.4 é descrito os requisitos e expectativas para o caso "Efectuar pedido de informação sobre reclamação". Para estes requisitos e expectativas do sistema, os agentes "Cidadão" e "Sistema de controlo" também compõem o modelo.

Tabela 6.4 - Requisitos e Expectativas do objectivo "Efectuar pedido de informação sobre reclamação".

	Efectuar pedido de informação sobre reclamação	**Descrição**
Expectativas	-Inserir dados sobre a reclamação.	-Para o pedido de informação sobre reclamações, o cidadão insere os dados referente a reclamação desejada.
Requisitos	-Verificar dados inseridos; -Mostrar informação sobre os dados inseridos.	-Após inseridos os dados da reclamação, estes são verificados no sistema e posteriormente é apresentada a informação sobre a reclamação.

A Figura 6.5 descreve estes requisitos e expectativas do sistema graficamente.

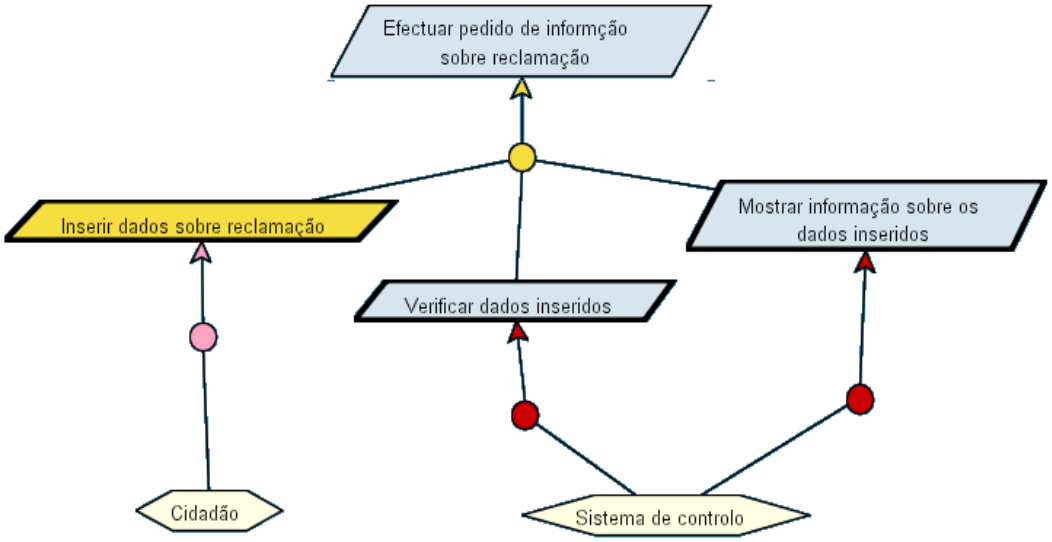


Figura 6.5 - Requisitos e Expectativa do objectivo "Efectuar pedido de informação sobre reclamação".

Para o caso do objectivo "Efectuar pedido de informação sobre unidades de saúde/ especialidades", estas encontram-se decompostas por "Efectuar listagem das especialidades por unidades de saúde" e "Efectuar listagem das unidades de saúde por especialidades". Estes objectivos também são compostos por alguns requisitos e expectativas do sistema que permitem que os mesmos sejam alcançados. Deste modo, na Tabela 6.5 é apresentado os requisitos e expectativas para cada sub-objectivo.

Tabela 6.5 - Expectativas e Requisitos dos objectivos " Efectuar pedido de informação sobre unidades de saúde/especialidades ".

	Efectuar listagem das unidades de saúde por especialidades	**Efectuar listagem das especialidades por unidades de saúde**
Expectativas	-Seleccionar opção "Unidades de saúde por especialidade" na página; -Seleccionar especialidade desejada.	-Seleccionar opção "Especialidades por unidade de saúde" na página; -Seleccionar unidade de saúde desejada.
Requisitos	-Mostrar lista de especialidades disponíveis no sistema; -Mostrar unidades de saúde disponíveis.	-Mostrar lista de unidades de saúde disponíveis no sistema; -Mostrar especialidades disponíveis.

Na Figura 6.6 é ilustrado os requisitos e as expectativas do objectivo "Efectuar listagem das unidades de saúde por especialidades".

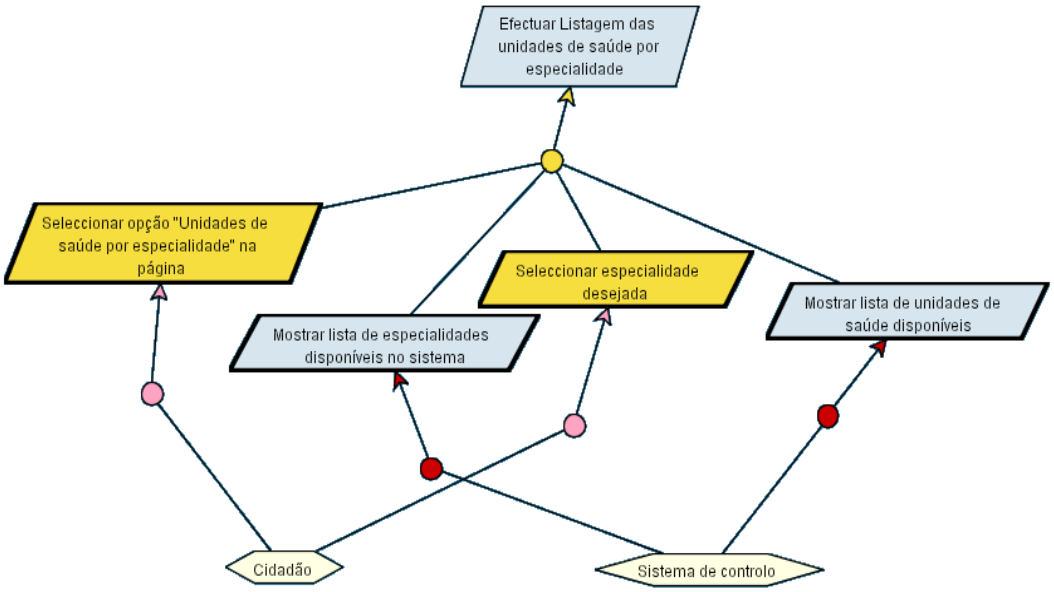


Figura 6.6 - Requisitos e Expectativas do objectivo "Efectuar listagem das unidades de saúde por especialidade".

A Figura 6.7 demonstra o objectivo "Efectuar listagem das especialidades por unidades de saúde " e sua decomposição.

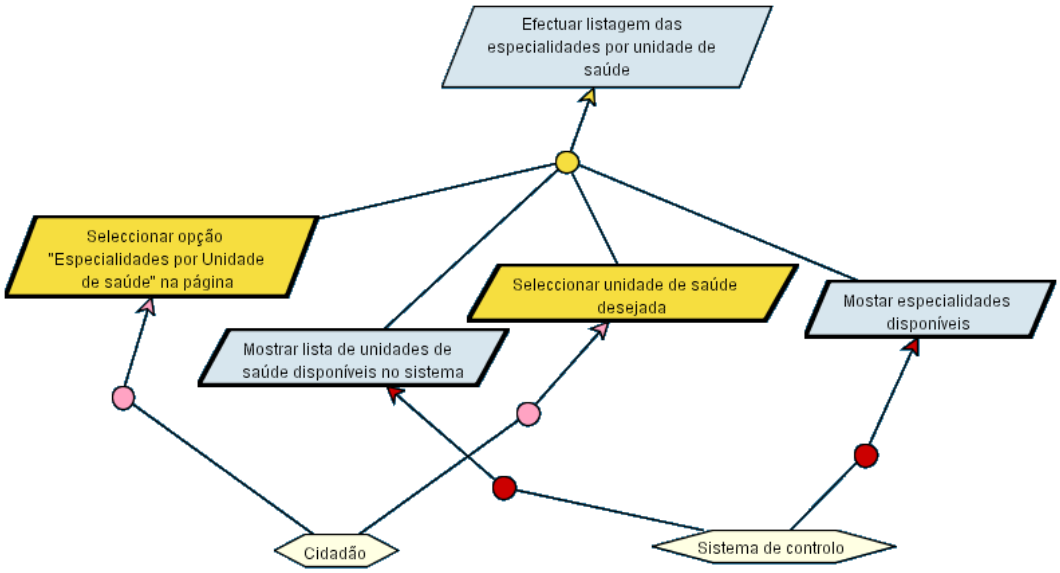


Figura 6.7 - Requisitos e Expectativas do objectivo "Efectuar listagem das especialidades por unidades de saúde".

O objectivo "Efectuar o envio das estatísticas ao director da unidade de saúde" é decomposto pelos requisitos "Gerar relatório de estatísticas das reclamações" e "Enviar relatório ao sistema". Este relatório é gerado pelo agente "Sistema de controlo" que posteriormente é

enviado ao director do sistema de saúde. Na Figura 6.8 é apresentada o modelo de objectivo relacionado com este objectivo.

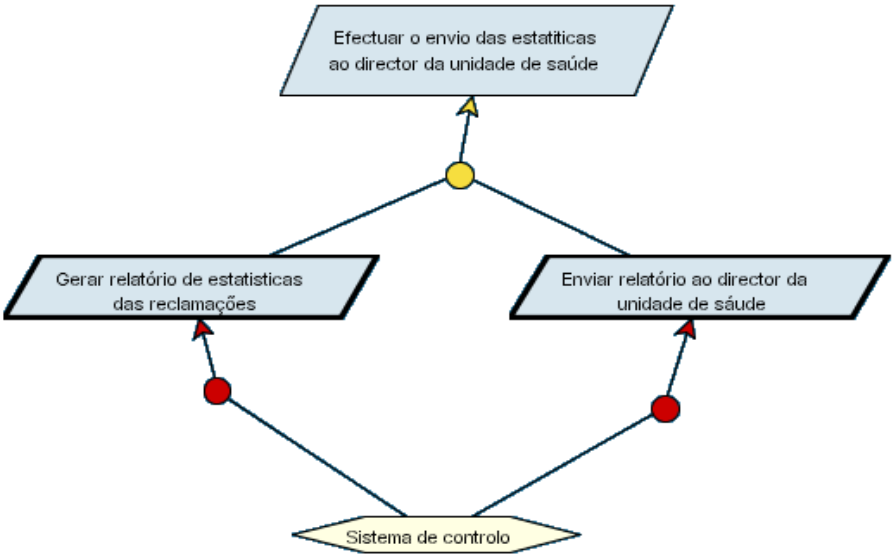


Figura 6.8 - Requisitos do objectivo "Efectuar o envio das estatísticas ao director da unidade de saúde".

O objectivo "Efectuar registo da reclamação no sistema" é decomposto pelos sub-objectivos "Efectuar registo da reclamação animal no sistema", "Efectuar registo da reclamação alimentar no sistema" e "Efectuar registo da reclamação diversa no sistema" como é ilustrado na Figura 6.9. Para cada caso, é seleccionada a opção de reclamação desejada (animal, alimentar ou diversa), fornecendo os dados necessários e a reclamação em si, que posteriormente serão guardados no sistema para serem enviados pelo assistente para o departamento que irá fazer a gestão da reclamação do cidadão. Na Tabela 6.6 são descritos os requisitos e as expectativas destes sub-objectivos.

Tabela 6.6 - Requisitos e expectativas para o objectivo "Efectuar registo da reclamação no sistema".

	Efectuar registo da reclamação animal no sistema	**Efectuar registo da reclamação alimentar no sistema**	**Efectuar registo da reclamação diversa no sistema**
Expectativas	-Seleccionar opção "Reclamação animal" na página; -Fornecer dados do cidadão; -Fornecer dados da reclamação; -Enviar a reclamação ao departamento indicado; -Gerir reclamação do cidadão.	-Seleccionar opção "Reclamação alimentar" na página; -Fornecer dados do cidadão; -Fornecer dados da reclamação; -Enviar a reclamação ao departamento indicado; -Gerir reclamação do cidadão.	-Seleccionar opção "Reclamação diversa" na página; -Fornecer dados do cidadão; -Fornecer dados da reclamação; -Enviar a reclamação ao departamento indicado; -Gerir reclamação do cidadão.
Requisitos	-Guardar os dados da reclamação.	-Guardar os dados da reclamação.	-Guardar os dados da reclamação.

Na Figura 6.9, são ilustrados os requisitos e as expectativas para os sub-objectivos citados na Tabela 6.6.

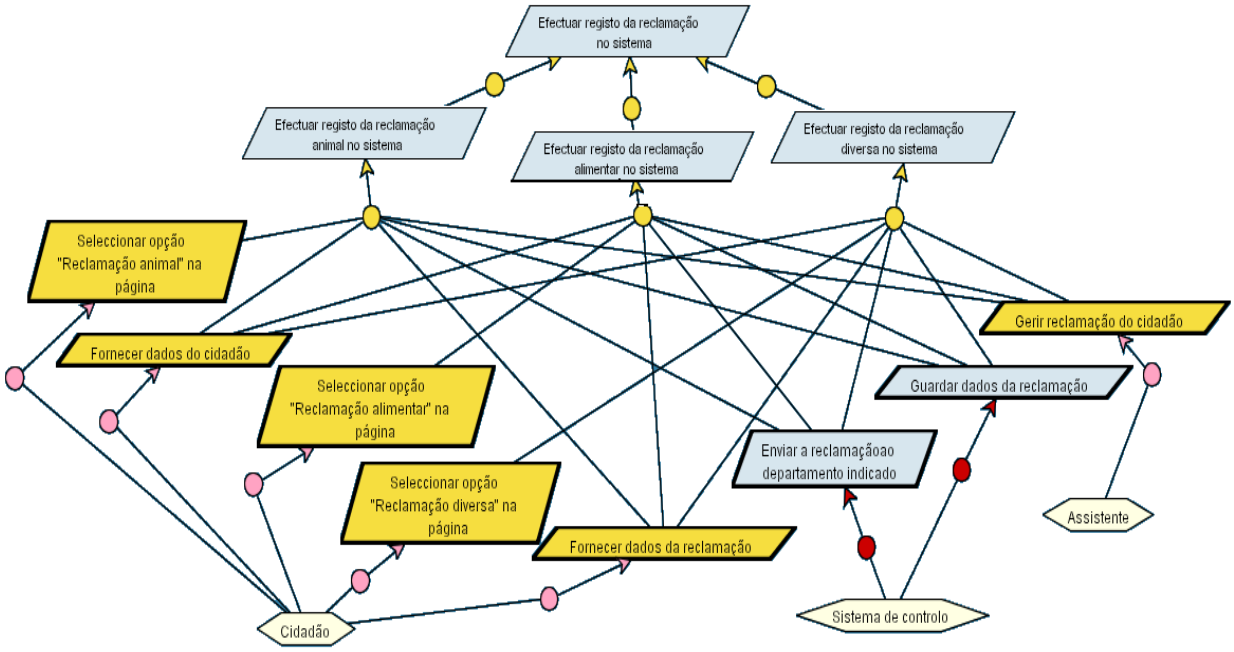


Figura 6.9 - Requisitos e Expectativas do objectivo "Efectuar registo da reclamação no sistema".

O objectivo "Efectuar a troca de informação com o SVS" é decomposto pelo requisito "Enviar informação da reclamação dos cidadãos" e pela expectativa "Receber informação do SVS". A Figura 6.10 ilustra o modelo de objectivo para este sub-objectivo.

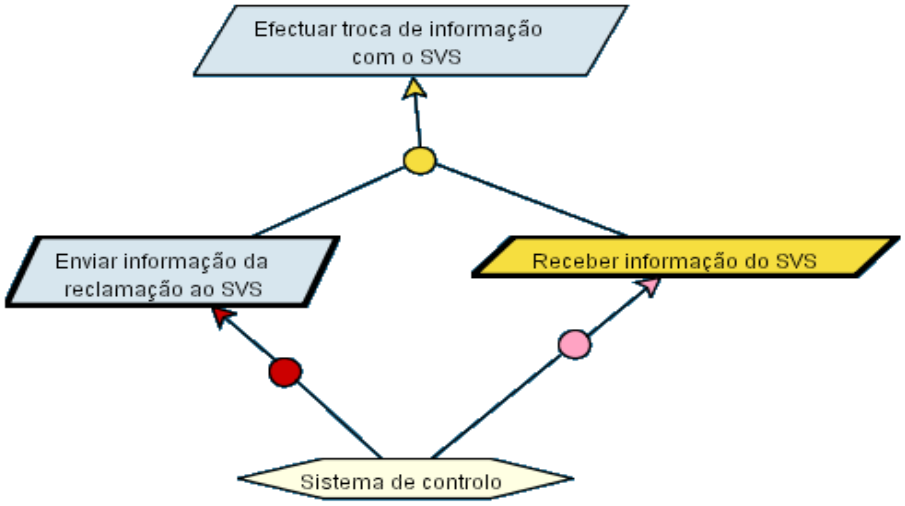


Figura 6.10 - Requisito e Expectativa do objectivo "Efectuar troca de informação com o SVS".

Depois da identificação dos objectivos do sistema e do desenvolvimento do modelo de objectivo estes são validados de modo a confirmar as decisões tomadas no sistema.

6.1.1.2 Identificação e especificação das *features* do sistema

Uma vez identificados e especificados os objectivos do sistema, seguidamente são obtidas as *features* do sistema de modo a especificar a linha de produtos de software.

Deste modo, utilizando as heurísticas para a identificação das *features* do sistema definidas na secção 5.2, na Tabela 6.7 são descritas as *features* que foram identificadas através dos objectivos e sub-objectivos do sistema de saúde pública.

Tabela 6.7 - Identificação das *features* a partir dos objectivos e sub-objectivos do sistema.

Objectivos	*Features*	Descrição
-Sistema de saúde pública	-Saúde pública	Feature "Saúde pública" que representa o sistema em desenvolvimento.
-Efectuar a registo do assistente	-Registo_Assistente -Dados_Assistente (requerida)	-*Feature* para o caso do registo dos assistentes que requer os dados do assistente para efectuar o registo.
-Efectuar pedido de informação ao sistema	-Informação -Reclamação (requerida) -Doenças (requerida) -Unidade de saúde (requerida) -Especialidade (requerida) -Campanha de vacinação (requerida)	-Uma vez que o sistema permite obter informação para a satisfação dos cidadãos, esta é uma propriedade importante para o sistema, visto que esta informação é referente a reclamação, doença, unidade de saúde, etc., que são requeridas.
-Efectuar o envio das estatísticas ao director da unidade de saúde	-Estatística -Reclamação (requerida)	-O sistema permite o envio de estatísticas ao director de uma unidade de saúde, deste modo, requer a informação referente as reclamações dos sistema para constar nestas estatísticas.
-Efectuar registo da reclamação no sistema.	- Reclamação -R. Animal (requerida) -R. Alimentar (requerida) -R. Diversa (requerida)	-*Feature* "reclamação" já identificada. Existem três tipos de reclamações (animal, alimentar e diversa) possíveis no sistema que permitem identificar as reclamações do sistema, deste modo estas são requeridas.
-Efectuar troca de informação com o SVS	-Informação	-Feature "Informação" já identificada
-Efectuar pedido de informação sobre campanha de vacinação	-Informação -Campanha de vacinação (requerida)	-*Features* já identificadas
-Efectuar pedido de informação sobre doença	-Informação -Doença (requerida)	-*Features* já identificadas
-Efectuar pedido de informação sobre reclamação	-Informação -Reclamação (requerida)	-*Features* já identificadas
-Efectuar pedido de informação sobre unidades de saúde por especialidade	-Informação -Unidade de saúde (requerida) -Especialidade (requerida)	-*Features* já identificadas
-Efectuar pedido de informação sobre	-Informação -Especialidade (requerida)	-*Features* já identificadas

especialidades por unidade de saúde	-Unidades de saúde (requerida)	
-Efectuar registo da reclamação animal no sistema	-Reclamação -R_Animal (requerida)	-*Features* já identificadas
-Efectuar registo da reclamação alimentar no sistema	-Reclamação -R_Alimentar (requerida)	-*Features* já identificadas
-Efectuar registo da reclamação diversa no sistema	-Reclamação -R_Diversa (requerida)	-*Features* já identificadas

Após a identificação das primeiras *features* do sistema, começa-se a produzir o modelo de *features* com a ajuda das heurísticas de desenvolvimento deste modelo também definidas na secção 5.2. Portanto, a Figura 6.11 ilustra a versão 1 do modelo de *features* a desenvolver.

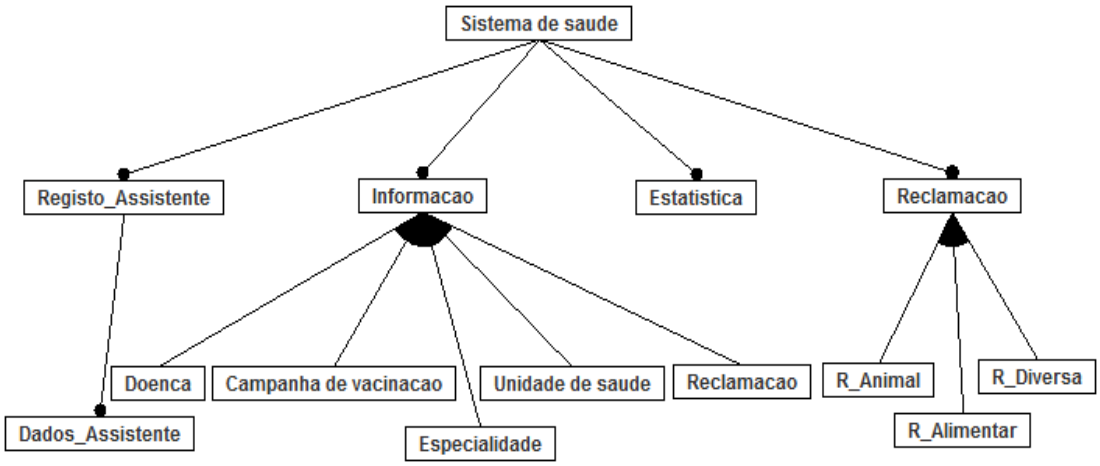


Figura 6.11 - Modelo de *features* incompleto - Versão 1.

Utilizando mais uma vez as heurísticas de obtenção das *features* do sistema, na Tabela 6.8 são descritas as *features* identificadas a partir dos requisitos que formam obtidas na secção 6.1.1.1. Nesta tabela encontram-se também algumas *features* já descritas anteriormente, permitindo reforçar o seu aparecimento no modelo de *features*.

Tabela 6.8 - Identificação de *features* a partir dos requisitos do sistema.

Requisitos	***Features***	**Descrição**
-Guardar dados do assistente	-Dados_Assistente	-*Feature* já identificada.
-Mostrar lista de doenças disponíveis	-Doença	-*Feature* já identificada.
-Mostrar informação da doença desejada	Informação -Doença (requerida)	-*Features* já identificadas.
-Mostrar lista de unidades de saúde disponíveis no sistema	-Unidade de saúde	- *Feature* já identificada. -
-Mostrar especialidades disponíveis	-Especialidade	-*Feature* já identificada.
-Verificar dados da reclamação inserida	-Dados_Reclamação	Ao pedir informação sobre uma reclamação, são inseridos dados que têm de se verificado no sistema para

		responde ao cidadão.
-Mostrar informação sobre os dados da reclamação inserida	-Informação -Dados_Reclamação	-*Feature*s já identificadas.
-Mostrar lista de unidades de saúde disponíveis	-Unidade de saúde	-*Feature* já identificada.
-Mostrar lista de especialidades disponíveis no sistema	-Especialidade	-*Feature* já identificada.
-Enviar a reclamação ao departamento indicado	-Reclamação	-*Feature* já identificada.
-Guardar dados da reclamação	-Dados_Reclamação	-*Feature* já identificada.
-Enviar informação da reclamação ao SVS	-Informação -Reclamação (requerida)	-*Features* já identificadas.
-Gerar relatório de estatística das reclamações	-Estatística -Reclamação (requerida)	-*Features* já identificadas
-Enviar relatório ao director da unidade de saúde	-Unidade de saúde	-*Feature* já identificada

A Figura 6.12 descreve a versão 2 do modelo de *features* que permite ilustrar as *features* identificadas através dos requisitos do sistema. Através das heurísticas de desenvolvimento do modelo de *features* pode-se constatar que a *feature* "Reclamação" é decomposta em duas sub-features, visto que com a obtenção da feature "Dados_Reclamação" é útil criar a *feature* suplementar "Tipo_R" que agrupa os três tipos de reclamação do sistema, permitindo estruturar o modelo.

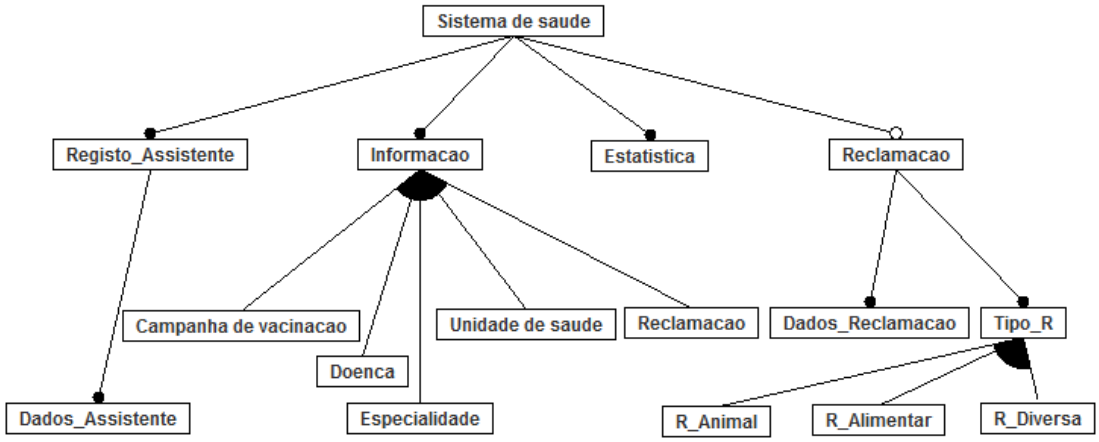


Figura 6.12 - Modelo de *features* incompleto - Versão 2.

As *features* identificadas a partir das expectativas de modelos de objectivos do sistema de saúde através das heurísticas para o efeito encontram-se descritas na Tabela 6.9.

Tabela 6.9 - Identificação de *features* a partir das expectativas do sistema.

Expectativas	*Features*	Descrição
-Fornecer dados do assistente	-Dados_Assistente	-*Feature* já identificada
-Receber informação do SVS	-Informação	-*Feature* já identificada
-Seleccionar opção "info-campanha de vacinação"	-Campanha de vacinação	-*Feature* já identificada
-Seleccionar campanha de vacinação desejada	-Campanha de vacinação	-*Feature* já identificada
-Seleccionar opção "Info-doença" na página	-Doença	-*Feature* já identificada
-Seleccionar doença desejada	-Doença	-*Feature* já identificada
-Seleccionar opção "Especialidades por unidade de saúde" na página	-Especialidade -Unidade de saúde (requerida)	-*Features* já identificadas
-Seleccionar unidade de saúde desejada	-Unidade de saúde	-*Feature* já identificada
-Inserir dados sobre reclamação	-Dados_Reclamação	-*Feature* já identificada
-Seleccionar opção "Unidades de saúde por especialidade" na página	-Unidade de saúde -Especialidade (requerida)	-*Features* já identificadas
-Seleccionar especialidade desejada	-Especialidade	-*Feature* já identificada
-Seleccionar opção "Reclamação animal" na página	-R_Animal	-*Feature* já identificada
-Seleccionar opção "Reclamação alimentar" na página	-R_Alimentar	-*Feature* já identificada
-Seleccionar opção "Reclamação diversa" na página	-R_Diversa	-*Feature* já identificada
-Fornecer dados do cidadão	-Dados_Cidadão	-Dados fornecidos pelo cidadão quando efectua a reclamação
-Fornecer dados da reclamação	-Dados_Reclamação	-*Feature* já identificada
-Gerir reclamação do cidadão	-Reclamação	-*Feature* já identificada

Após a obtenção das *features* do sistema a partir das expectativas, é ilustrada na Figura 6.13 o modelo de *features* com as novas features identificadas.

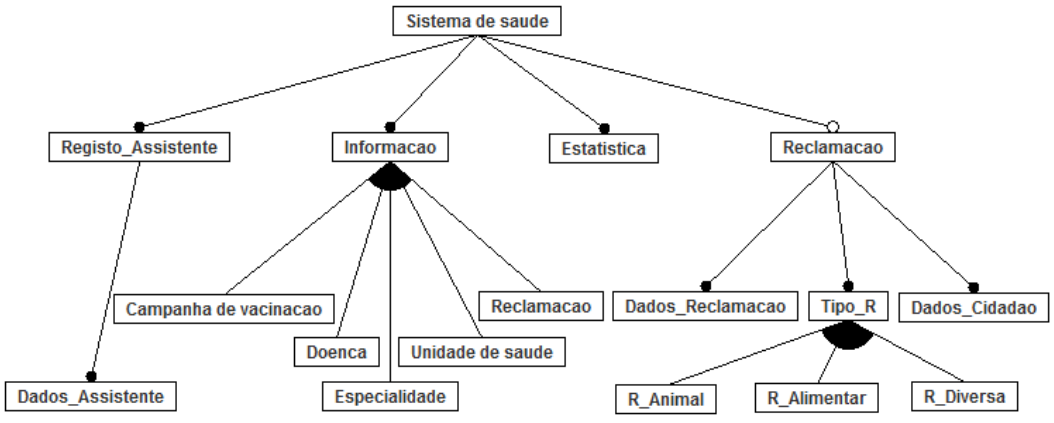


Figura 6.13 - Modelo de *features* incompleto - Versão 3.

Uma vez identificadas as *features* e efectuada a construção do modelo de *features* incompleto, continua-se a definir as variabilidades e interacções entre as *features*. Deste modo a Figura 6.14, descreve o modelo de *features* final do sistema de saúde pública.

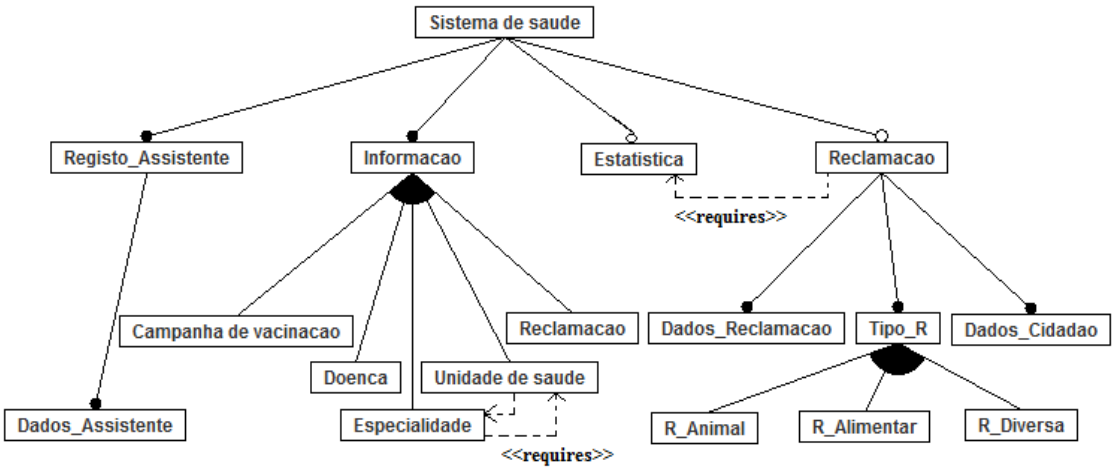


Figura 6.14 - Modelo de *features* do sistema de saúde.

A fase seguinte depois da identificação e especificação das *features* do sistema é a validação das opções tomadas, isto é, validação das *features* do sistema e suas propriedades, é efectuada uma revisão minuciosa de modo a saber que o que se desenvolveu corresponde com as especificações desejadas. Estas revisões são efectuadas entre os clientes e os engenheiros de sistema.

A configuração da LPS para uma aplicação é tratada na secção seguinte na Engenharia de Aplicação.

6.1.2 Engenharia da Aplicação

A nível da engenharia da aplicação é configurado o sistema para uma determinada instância de uma aplicação.

6.1.2.1 Configuração do modelo de objectivos do sistema

De modo a apresentar uma configuração possível do modelo de objectivos, utilizou-se o caso do pedido de informação sobre doenças e a reclamação do tipo animal. Na Figura 6.15 - Configuração dos objectivos e sub-objectivos do sistema.é ilustrada a configuração dos objectivos e sub-objectivos do sistema para estes casos referidos.

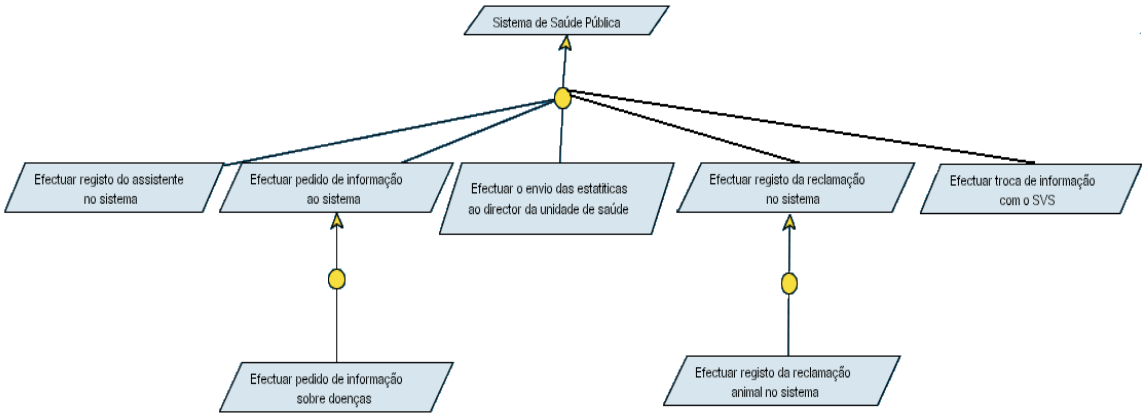


Figura 6.15 - Configuração dos objectivos e sub-objectivos do sistema.

Para o sub-objectivo "Efectuar pedido de informação sobre doenças" a descrição efectuada na Figura 6.3 é mantida sem alterações assim como para os restantes objectivos da Figura 6.15. Para o caso de "Efectuar registo da reclamação animal no sistema" tem-se a configuração apresentada na Figura 6.16.

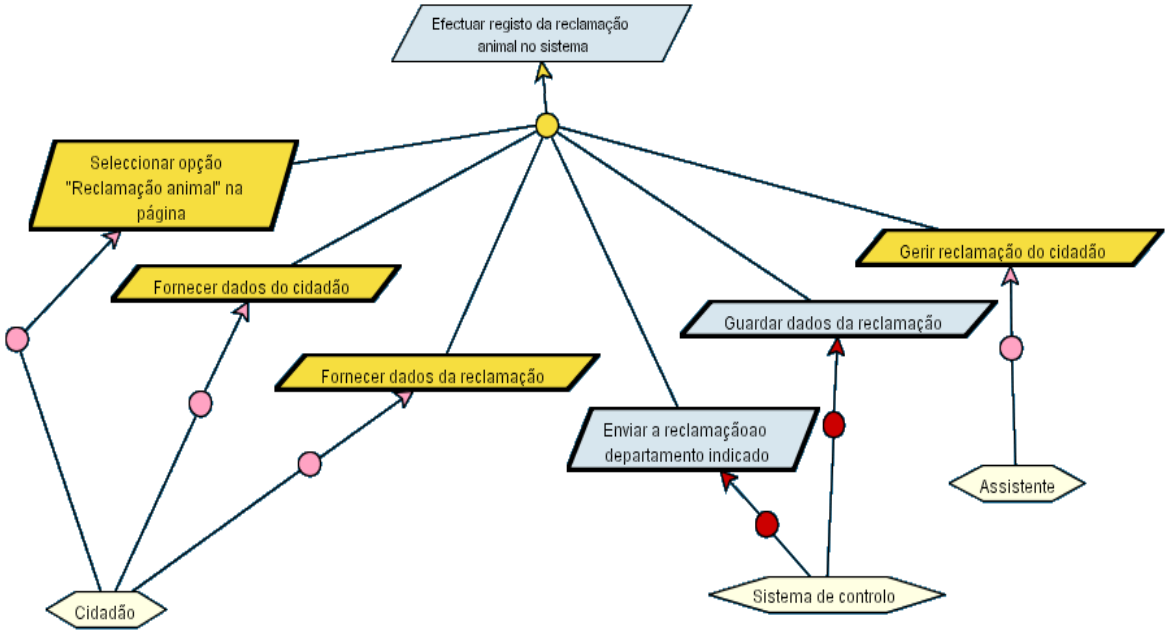


Figura 6.16 - Configuração do sub-objectivo "Efectuar registo da reclamação animal no sistema".

Na secção seguinte é efectuada a configuração do modelo de *features* do sistema.

6.1.2.2 Configuração do modelo de *features* do sistema

Depois de configurar o modelo de objectivos do sistema para a aplicação desejada, seguidamente é efectuada a configuração para o modelo de *features* do sistema. Esta configuração é demonstrada na Figura 6.17.

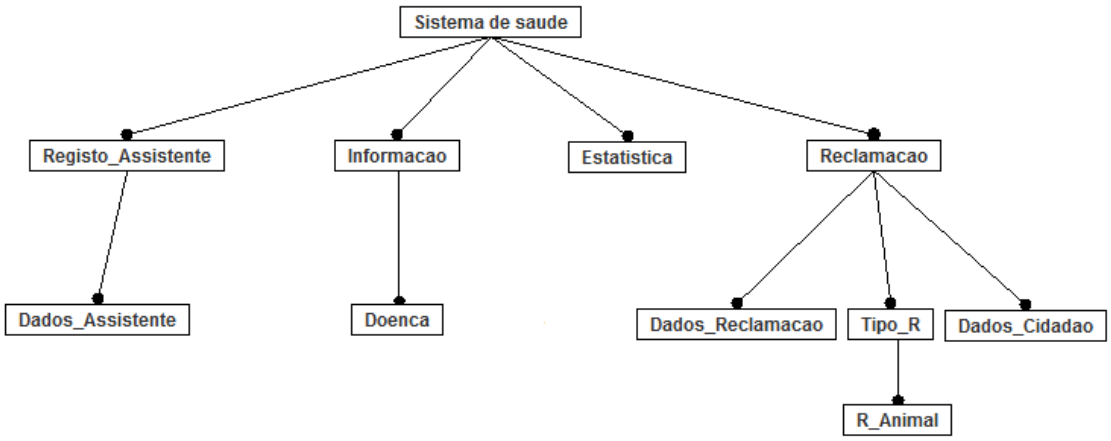


Figura 6.17 - Configuração do modelo de *features*.

Configurando o modelo de *features* em árvore de directório, temos a figura seguinte:

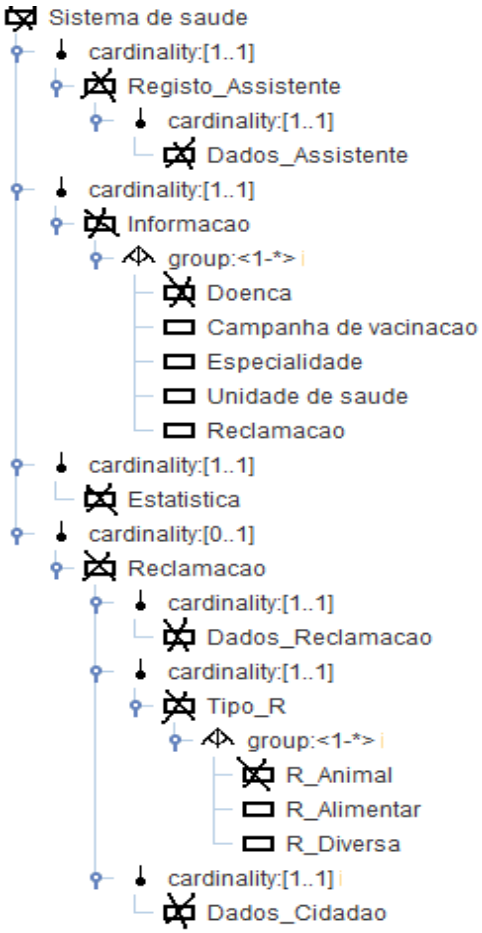


Figura 6.18 - Configuração do sistema para uma instância em forma de árvore de directório.

Seguidamente é efectuada uma comparação da abordagem LPS-KAOS com outras abordagens.

6.2 Comparação da abordagem LPS-KAOS com outras abordagens

Neste capítulo, a abordagem LPS-KAOS será alvo de comparação com algumas abordagens referidas nos trabalhos relacionados. Para tal, foi definido um conjunto de critérios que foram utilizados em [42]. Deste modo são listados os seguintes critérios de comparação:

- Modelação de requisitos;
- Modelação das propriedades comuns e variáveis;
- Modelação do domínio;
- Modelação de *features*;

- Área do domínio;
- Suporte para a engenharia da aplicação.

Na modelação dos requisitos são definidas as actividades para obter os requisitos funcionais e arquitecturais para a linha de produto, e as suas dependências entre elas. Isto permite incluir também o mapeamento das restrições específicas para os requisitos.

Para a modelação das propriedades comuns e variáveis são descritas as actividades para identificar as propriedades comuns e variáveis entre os requisitos. Permite incluir a separação de requisitos que são válidos para todo o domínio, daqueles que são válidos apenas em casos especiais, por exemplo para uma variante específica do produto. Esta actividade é fortemente relacionada para a modelação do domínio e das *features*.

Na modelação do domínio são especificadas os domínios e as dependências entre as actividades que permitem esta especificação.

A modelação das *features* permite identificar, estudar e descrever as features relevantes num dado domínio. O objectivo da modelação de *features* é expressar relações entre *features*, propriedades das *features* e/ou estruturas das *features*. Uma característica importante da modelação é as propriedades comuns e variáveis. Também permitem a configuração e interacção das *features*. Um propósito da modelação de *features* é ajudar a estruturar os requisitos e definir as variantes permitidas em uma LPS.

No suporte para a engenharia de aplicação é identificada as variações entre os requisitos de uma instância da LPS e os requisitos da LPS.

Para ajudar nas decisões dos critérios, um conjunto de possibilidades foi determinado:

- "Sim", se o critério é satisfeito;
- "Não", se o critério não é satisfeito;
- "Não Especificado", se o critério não é especificado no documento.

6.2.1 Aplicação do critério

Neste capítulo será efectuada a comparação da nossa abordagem (LPS-KAOS) com as outras abordagens descritas no capítulo dos trabalhos relacionados. A Tabela 6.10 é ilustrada esta comparação usando os critérios definidos.

Tabela 6.10 - Comparação entre LPS-KAOS e outras abordagens.

Critério/ Abordagens	**LPS-KAOS**	**Casos de Uso e LPS**	**Modelo de *features* e EROO**	**MATA e LPS**	**I* aspectual e LPS**
-Modelação de requisitos (Orientado a)	Sim (Objectivos)	Sim (Casos de Uso)	Sim (Objectivos)	Sim (Objectos)	Sim (Objectivos e aspectos)
-Modelação das variabilidades	Sim	Sim	Sim	Sim	Sim
-Modelação do domínio (Modelo de)	Sim (Objectivos do KAOS)	Sim (Casos de uso)	Sim (Objectivos genérico)	Sim (Objectos)	Sim (Objectivos do i*)
-Modelação das *features*	Sim	Sim	Sim	Sim	Sim
-Suporte para a engenharia da aplicação	Sim	Não	Não Especificado	Não Especificado	Sim

Sendo a abordagem LPS-KAOS uma abordagem orientada a objectivos e o Casos de Uso e LPS, uma abordagem orientada a casos de uso, em que ambas foram especificadas para incorporar as LPSs, faz sentido efectuar uma comparação entre as duas abordagens. Como se pode constatar na tabela, ambas as abordagens suportam quase todos os critérios definidos, diferenciando apenas no suporte da Engenharia da Aplicação que o caso de usos e LPS não suporta, isto porque o documento descreve apenas uma maneira de representar as *features* através dos casos de uso, sem mostrar como poderiam ser configurados para uma aplicação especifica.

Seguidamente, é efectuada a comparação entre a nossa abordagem e o modelo de *features* e EROO, visto que as abordagens têm o mesmo paradigma orientado a objectivos e focam as LPSs. Como se pode constactar na tabela 6.10, para os critérios definidos apenas o suporte

para a engenharia da aplicação não é satisfeito, visto que o documento que descreve esta abordagem (modelo de *features* e EROO) não especifica esta propriedade.

Para a abordagem MATA e LPS, verifica-se mais uma vez que o suporte da engenharia da aplicação não é satisfeito.

A abordagem I* Aspectual e LPS satisfaz todos os critérios propostos neste documento. Uma vez que a abordagem permite fazer o modelo de objectivos recorrendo ao modelo de *features*, esta possui algumas limitações, isto é, consideram o que é opcional no modelo de *features* e representam como um elemento aspectual no modelo de objectivos, o que nem sempre se verifica.

6.3 Resumo

Neste capítulo foi apresentado um caso de estudo para desenvolver um sistema de saúde pública denominado observador de saúde (*Health – Watcher*), que é baseado num sistema da indústria real. Este sistema foi desenvolvido com a nossa abordagem LPS-KAOS, de modo a ilustrar mais uma vez a mesma.

Para além do caso de estudo descrito, foi também efectuada uma comparação da nossa abordagem com outras definidas no capítulo dos trabalhos relacionados.

7 Conclusão

As exigências para obtenção de software que satisfaça as necessidades dos clientes têm aumentado, implicando uma maior preocupação por parte das empresas para fornecer maior variabilidade de produtos. Visto que as empresas desenvolviam produtos da mesma família separados em cada projecto de software, tinham de voltar a desenvolver partes já elaboradas em outros sistemas tornando o desenvolvimento menos eficiente. Deste modo, surge as LPSs que utilizam a propriedade de reutilização de componentes de software e proporcionam vários benefícios, tais como o aumento da produtividade de vários produtos semelhantes, melhoria da qualidade do software principalmente a nível de robustez, redução de custos na produção de uma família de produtos, menor tempo de desenvolvimento, entre outros. As LPSs fornecem às empresas uma capacidade de gestão de forma integrada no desenvolvimento de produtos e a sua implementação de acordo com as necessidades do cliente. Actualmente o uso das linhas de produtos de software tem-se acentuado cada vez mais e algumas abordagens orientadas a objectivos têm sido propostas para desenvolver LPS.

Nesta tese de mestrado foi proposta uma abordagem (LPS-KAOS) orientada a objectivos que permite especificar as linhas de produtos de software. Esta abordagem é composta por dois processos principais: A Engenharia de Domínio e a Engenharia da Aplicação. O processo da Engenharia de Domínio é subdividido em duas actividades: Identificação e especificação dos objectivos do sistema que subdivide-se em modelação dos objectivos do sistema e em Validar os Objectivos, Requisitos, Expectativas e Agentes do sistema; e a Identificação e especificação das *features* do sistema, que se encontra subdividido na modelação das *features* do sistema, Identificar as variabilidades e as interacções entre as *features* do sistema e validar as features do sistema e as suas respectivas interacções. O processo de Engenharia da Aplicação é composto também pela actividade de especificação dos modelos do sistema para

uma aplicação que subdivide-se na configuração do modelo de objectivos e do modelo de *features* do sistema para uma aplicação. A abordagem foi aplicada em dois exemplos práticos: Sistema de parqueamento e sistema de saúde pública.

7.1 Contribuições

A abordagem descrita nesta dissertação é uma abordagem para as LPSs que integra modelação de *features*. Deste modo, na abordagem foram definidas um conjunto de heurísticas que permitem a sistematização da identificação das *features* para um sistema e também permitem metodicamente o desenvolvimento do modelo de *features*.

Com a realização desta dissertação vai permite realçar os resultados produzidos por uma abordagem de requisitos orientada a objectivos, introduzindo um mecanismo mais eficiente da modularização, tendo como alvo apontado um requisito de sistema compreensível e mais envolvente.

Portanto, a maior contribuição desta dissertação é beneficiar a ELPS com a EROO e também expandir o horizonte da abordagem KAOS para que esta seja usada na Engenharia de Linhas de Produtos de Software.

7.2 Limitações

No âmbito do objectivo desta dissertação podemos apontar as seguintes limitações ao trabalho realizado:

- Não contempla os outros modelos da abordagem KAOS, os conflitos e obstáculos, e também os *softgoals*;
- Falta de uma ferramenta que suporta a abordagem LPS-KAOS, de modo a encorajar a sua utilização.

7.3 Trabalhos futuros

Como foi descrito neste documento, apenas o modelo de objectivos da abordagem KAOS foi contemplado para a especificação da LPS. Deste modo, para trabalho futuro de modo a tornar a abordagem mais completa, seria a integração dos outros modelos a qual a abordagem KAOS é composta e também incorporação dos obstáculos e conflitos. Apesar do modelo de *features* não tratar dos requisitos não-funcionais, os *softgoals* da abordagem KAOS poderiam dar um contributo na modelação de *features* de um sistema.

Para trabalho futuro importa também potenciar o uso de uma ferramenta de suporte para a abordagem LPS-KAOS, capaz de identificar as *features* a partir dos objectivos de um sistema.

Bibliografia

1. Kotonia G, Sommerville I. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, 1998.

2. Sommerville I, Sawyer P. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, 1997.

3. Bertrand P, Darimont R, Delor E, Massonet P, Lamsweerde A. GRAIL/KAOS: an environment for goal driven requirements engineering, *Proceedings ICSE'98 - 20th International Conference on Software Engineering*: Kyoto, 1998.

4. Lamsweerde A. Goal-Oriented Requirements Engineering: A Guided Tour, *In RE'01 - 5th IEEE International Symposium on Requirements Engineering*: Toronto, 2001.

5. Lapouchnian A. Goal-Oriented Requirements Engineering: An Overview of the Current Research. University of Toronto, 2005.

6. Finkelstein A, Kramer J, Nuseibeh B, Finkelstein L, Goedicke M. Viewpoints: A framework for integrating multiple perspectives in system development*Int. J. Software Engineering Knowledge* 1992;31-58.

7. Rumbaugh J, Booch G, Jacobson I. *The Unified Modeling Language Reference Manual*. Addison Wesley: Reading MA, 2004.

8. Gomaa H. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison Wesley: Reading MA, 2000.

9. Clements P, Northrop L. *Software Product Lines: Practices and Patterns*. Addison-Wesley: Boston, MA, USA, 2007.

10. Pohl K, Böckle G, Van Der Linder F. *Software Product Line Engineering Foundations, Principles, and Techniques*. Springer: New York, 2005.

11. Kuloor C, Eberlein A. Requirements Engineering for Software Product Lines. University of Calgary: Canada, 2002.

12. Lamsweerde A, Darimont R, Letier E. Managing Conflicts in Goal-Driven Requirements Engineering *IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development*, 1998.

13. Kavakli E. Modeling organizational goals: analysis of current methods, *Proceedings of the 2004 ACM symposium on Applied Computing*: Nicosia, Cyprus, 2004;1339-1343.

14. Lamsweerde A. KAOS Tutorial. Cediti, 2003.

15. Yu E. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering, *Proc. RE-97 - 3rd Int. Symp. on Requirements Engineering, Annapolis*: Annapolis, 1997;226 - 235.

16. GRL web site, http://www.cs.toronto.edu/km/GRL/, accessed in April.

17. Mussbacher G, Amoyt D, Araujo J, Moreira A, Weiss M. Visualizing Aspect-Oriented Goal Models with AoGRL, *Second International Requirements Engineering Visualization*, 2007.

18. Yu Y, Leite J, Mylopoulos J. From goals to aspects: discovering aspects from requirements goal models, *Proceedings of the 12th IEEE International Requirements Engineering Conference*: Kyoto, Japan, 2004;September.

19. Chung L, Nixon BA, Yu E, Mylopoulos J. Non-Functional Requirements in Software Engineering. Kluwer Academic: Boston, 2000.

20. Lamsweerde A, Letier E. From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering, *Proc. Radical Innovations of Software and Systems Engineering*, 2003.

21. Objectiver web site, http://www.objectiver.com, accessed in April, 2008.

22. Moodle web site, www.moodle.fct.unl.pt, Acetactos das aulas de ERDS, FCT-UNL-DI, Professor João araújo, accessed in april, 2008.

23. Moodle web site, www.moodle.fct.unl.pt, trababalho final de ERDS, FCT-UNL-DI, Professor João Araújo, accessed in april, 2008.

24. Yu E. Modeling Strategic Relationships for process Reengineering: Toronto, 1995.

25. Silva L, Sampaio J. Generating Requirements Views: A Transformation-Driven Approach, *In: Proc. of 3rd Workshop on Software Evolution through Transformations*: Natal, Rio de janeiro, 2006.

26. Amyot D. Introduction to the User Requirements Notation: Learning by Example. Computer Networks. 2003; 42(3): 285 - 301

27. Griss ML. Implementing Product-Line Features with Component Reuse*Software Reuse: Advances in Software Reusability: 6th International Conference*: Vienna-Austria, 2000.

28. Weiss D, Lai C. Software Product-Line Engineering - a family-based software development process. Addison-Wesley: Reading MA, 1999.

29. Pohl K, Metzger A. Variability Management in Software Product Line Engineering, *Proceedings of the 28th international conference on Software engineering*, 2006;1046-1050.

30. Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Carnegie Mellon University, Software Engineering Institute, 1990.

31. Kang KC, Kim S, Lee J, Kim K. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures, *Annals of Software Engineering* 1998;143-168.

32. Riebisch M. Towards a More Precise Definition of Feature Models, *Position Paper In: M. Riebisch, J. O. Coplien, D, Streitferdt (Eds.): Modelling Variability for Object-Oriented Product Lines*. BookOnDemand Publ. Co: Norderstedt, 2003;64-76.

33. Czarnecki K, Helsen S, Eisenecker U. Staged Configuration Using Feature Models, *Software Product Lines: Third International Conference SPLC 2004*: Boston MA USA, 2004;266-283.

34. Captain Feature web site, https://sourceforge.net/projects/captainfeature/, accessed in june, 2008.

35. Gomaa H. *Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures*. Addison-Wesley, 2004.

36. Silva C, Alencar F, Araújo J, Moreira A, CASTRO JFB. Tailoring an Aspectual Goal-Oriented Approach to Model Features*The 20th International Conference on Software Engineering and Knowledge Engineering* San Francisco Bay, USA, 2008.

37. Yu Y, do Prado Leite JCS, Lapouchnian A, Mylopoulos J. Configuring features with stakeholder goals. *Proceedings of the 2008 ACM symposium on Applied computing* 2008: 645-649

38. Jayaraman P, Whittle J, Elkhodary AM, Gomaa H. Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis. *LECTURE NOTES IN COMPUTER SCIENCE* 2007; 4735: 151

39. Czarnecki K, Antkiewicz M. Mapping Features to Models: A Template Approach Based on Superimposed Variants, 2005;422-437.

40. Matulevicius R, Heymans P. Analysis of KAOS Meta-model. Namur University: Computer Science Department: Belgiun, 2005.

41. Soares S, Borba P, Laureano E. Distribution and Persistence as Aspects. Software: Practice & Experience. 2006; 36(6)

42. Alférez M, Moreira A, Araújo J, Amaral V. AMPLE Aspect-Oriented, Model-Driven, Product Line engineering Specific Targeted Research project: I. *Lancaster University (United Kingdom)* june, 2007